

北京邮电大学

本科毕业设计（论文）



题目：一种基于工作量的 Serverless 计算自动伸缩算法的设计
与实现

姓 名 张梓靖
学 院 计算机学院
专 业 智能科学与技术
班 级 2019211315
学 号 2019211379
班内序号 27
指导教师 王纯

2023 年 5 月

北京邮电大学

本科毕业设计（论文）诚信声明

本人声明所呈交的毕业设计（论文），题目《一种基于工作量的 Serverless 计算自动伸缩算法的实现与实现》是本人在指导教师的指导下，独立进行研究工作所取得的成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京邮电大学或其他教育机构的学位或证书而使用过的材料。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名：_____张梓靖_____ 日期：_____2023 年 5 月 23 日_____

关于论文使用授权的说明

本人完全了解并同意北京邮电大学有关保留、使用学位论文的规定，即：北京邮电大学拥有以下关于学位论文的无偿使用权，具体包括：学校有权保留并向国家有关部门或机构送交学位论文，有权允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，有权允许采用影印、缩印或其它复制手段保存。汇编学位论文，将学位论文的全部或部分内容编入有关数据库进行检索。（保密的学位论文在解密后遵守此规定）

本人签名：_____张梓靖_____ 日期：_____2023 年 5 月 23 日_____

导师签名：_____王电_____ 日期：_____2023 年 5 月 23 日_____

北 京 邮 电 大 学

本科毕业设计（论文）任务书

学院	计算机学院（国家示范性软件学院）	专业	智能科学与技术	班级	2019211315					
学生姓名	张梓靖	学号	2019211379	班内序号	27					
指导教师姓名	王纯	所在单位	计算机学院（国家示范性软件学院）	职称	高级工程师					
设计(论文)题目	（中文）一种基于工作量的 Serverless 计算自动伸缩算法的设计与实现									
	（英文）Design and Implementation of Workload-based Auto-scaling Algorithm for Serverless Computing									
题目分类	工程实践类 <input type="checkbox"/> 研究设计类 <input checked="" type="checkbox"/> 理论分析类 <input type="checkbox"/>									
题目来源	题目是否来源于科研项目 是 <input type="checkbox"/> 否 <input checked="" type="checkbox"/>									
	科研项目名称:									
	科研项目负责人:									
<p>主要任务及目标:</p> <ul style="list-style-type: none">1、学习 Kubernetes 容器编排系统的知识，并在虚拟机中搭建一个 Kubernetes 集群，熟悉 Kubernetes 的基本操作，并基于该集群环境进行后续实验。重点学习和使用 Kubernetes 的工作负载资源 Deployment 和 HPA 控制器，并理解 HPA 控制器的基于阈值的自动水平伸缩工作负载实例的算法。2、学习 Serverless 计算的知识，并在搭建的 Kubernetes 集群中尝试使用 Serverless 框架 OpenFaaS。3、根据 Serverless 工作负载流量的突发性，设计一个 Serverless 服务，其流量按照一定规律随时间而变化，并且可通过 OpenFaaS 框架部署到搭建好的 Kubernetes 集群。以设计的 Serverless 服务为示例应用，设计实现用来预测 Serverless 工作负载请求流量的时间序列预测算法。4、以设计的 Serverless 服务为示例应用，设计实现根据预测流量值的自动水平伸缩工作负载实例的算法。5、将算法部署到 Kubernetes 集群，与 OpenFaaS 框架进行交互。										
<p>主要内容:</p> <ol style="list-style-type: none">1、学习和使用 Kubernetes 容器编排系统。2、学习并掌握 Serverless 计算的开源框架 OpenFaaS。3、调研并使用主流时间序列预测方法，根据历史数据来预测 Serverless 工作负载的未来请求流量情况，以进行工作负载实例数量的自动水平伸缩。4、部署到 Kubernetes 集群进行验证。										

主要参考文献:

- [1]Kubernetes Authoritative guide version 4, author: Zheng Gong, Zhihui Wu, Xiulong Cui, Jianyong Yan.
- [2]Docker technology introduction, author: Baohua Yang.
- [3]Kubernetes docs: <https://kubernetes.io/docs/home/>
- [4]OpenFaaS docs: <https://docs.openfaas.com/>
- [5]Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti and et al., "Cloud programming simplified: A berkeley view on serverless computing," arXiv preprint arXiv:1902.03383, 2019.
- [6]Laszlo Toka, Gergely Dobreff, Balazs Fodor and Balazs Sonkoly, "Adaptive AI-based auto-scaling for Kubernetes," in 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID). IEEE, 2020, pp. 559-608.

进度安排:

- 1、学习和使用 Kubernetes 容器编排系统, 在 Kubernetes 集群中尝试使用 Serverless 框架 OpenFaaS。2022.11-12
- 2、基于 OpenFaaS 设计一个 Serverless 示例应用, 设计实现用来预测 Serverless 工作负载请求流量的时间序列预测算法, 并设计实现根据预测流量值的自动水平伸缩工作负载实例的算法。2023.1-3
- 3、将设计的算法部署到 Kubernetes 集群上进行实际测试, 获取实验结果, 开始撰写论文。2023.4-5
- 4、修改完善论文, 进行答辩。2023.6

指导教师签字



日期

2022 年 12 月 9 日

北京邮电大学计算机学院

本科生毕业设计（论文）成绩评定表

学生姓名	张梓靖		所在学院	计算机学院（国家示范性软件学院）			
学号	2019211379	专业	智能科学与技术		班级	2019211315	
论文题目	一种基于工作量的 Serverless 计算自动伸缩算法的设计与实现						
	Design and Implementation of Workload-based Auto-scaling Algorithm for Serverless Computing						
指导教师姓名	王纯	指导教师职称	高级工程师		指导教师单位	北京邮电大学	
中期检查小组评分	(满分 25 分): 23 中期检查小组组长签字: 王纯 检查日期: 4 月 14 日						
指导教师评分	评价内容 (BY: 毕业要求)	具体要求			分值	评分 指导教师 复议	
	调研论证 (BY4、5)	能独立查阅文献和从事相关调研; 能正确翻译外文资料; 有收集、加工各种信息及获取新知识的能力和自学能力。			5	4	
	方案设计 (BY3、4、5、11)	能独立提出符合选题的可行性研究方案、实验方案、设计方案, 独立进行实验(如安装、调试、操作)和研究方案论证; 能够使用相关领域工具或平台完成系统或模块的设计和开发; 在设计环节中, 能够应用工程化思想, 合理评估系统成本, 并考虑社会、健康、安全、法律、文化以及环境等因素; 理解和评价工程实践对环境、社会可持续发展的影响。			5	4	
	能力水平 (BY4、5)	能综合运用所学知识和技能去分析与解决毕业设计(论文)过程中遇到的实际问题; 能正确处理实验数据; 能对课题进行理论分析, 得出有价值的结论。			5	5	
	学习态度 (BY10、11)	认真、勤奋、努力、诚实、严格遵守纪律, 按期饱满完成规定的任务。			3	3	
	设计(论文)水平 (BY3、4、11)	文题相符、综述简练完整, 有见解; 立论正确, 论述充分, 结论严谨合理; 实验正确, 分析处理科学; 文字通顺, 技术用语准确, 符合工程管理规范; 设计(论文)有理论价值和应用价值。			5	4	
	文本规范 (BY10)	装订顺序正确, 字体字号等与基本规范相符, 符号统一, 编号齐全, 图表完备、整洁、正确。			2	2	
	指导教师评分合计(满分 25 分): 22						
	评语: 论文 serverless 计算平台的自动伸缩机制为选题, 符合当前技术发展趋势, 具有较好的实用意义。论文对于 Serverless 计算模式下的自动伸缩问题进行了较充分的研究分析, 设计实现了基于 ARIMA 模型的自动伸缩算法, 通过实验验证了算法的有效性和优化效果; 并在算法部署中实现了基于 Prometheus 的监控方案, 验证了可用性。张梓靖同学在毕业设计中能较好地运用所学理论和有关专业知识, 论文结构合理, 表述清晰准确, 达到本科毕设要求, 同意答辩。						
	指导教师签字: 王纯 日期: 年 月 日						

复 议

☐是 ☐否 复议评分合计：

复议人签字：

复议日期：

复议有权限修改指导教师评分，选择复议后指导教师评分将由复议评分替换

本科生毕业设计（论文）答辩成绩评定标准				
答 辩 小 组 成 绩 评 定	评价内容 <small>(BY:毕业要求)</small>	具体要求	分值	评分
	选题 <small>(BY4)</small>	符合专业培养目标，符合社会实际、结合工程实际，难易适度，体现新颖性、综合性。	5	4
	设计（论文） 质量水平 <small>(BY3、4、5、11)</small>	全面完成任务书中规定的各项要求，文题相符，工作量饱满，写作规范，达到综合训练的要求，有理论价值和应用价值。能够使用相关领域工具或平台完成系统或模块的设计和开发；在设计环节中，能够应用工程化的思想，合理评估系统成本，并考虑社会、健康、安全、法律、文化以及环境等因素；理解和评价工程实践对环境、社会可持续发展的影响。	20	16.5
	答辩准备 <small>(BY10)</small>	准备充分；有简洁、清晰、美观的演示文稿；准时到场。	5	4
	内容陈述 <small>(BY3、4、5、10)</small>	语言表达简洁、流利、清楚、准确，思路清晰，重点突出，逻辑性强，概念清楚，论点正确；实验方法科学，分析归纳合理；结论严谨；表现出对毕业设计（论文）内容掌握透彻。	15	12.5
	回答问题 <small>(BY3、10)</small>	回答问题准确、有深度、有理论根据、基本概念清晰。	5	4
	答辩小组评分合计（满分 50 分）		41	
	意见： 论文设计与实现了基于工作量的 Serverless 计算自动伸缩算法，进行较为合理的功能划分，包括算法设计和部署实现。算法主要设计了一种基于 ARIMA 模型的自动伸缩算法，实现并对比了其他常见预测模型(如 MA、VAR、Prophet 模型)，实验验证取得了较好的效果。论文总体技术方案合理，表述通顺，满足毕设要求。通过答辩。			
	答辩小组组长签字：王玉坤 2023 年 5 月 30 日			
答辩小组成员：李伟 张强				
学 院 意 见	最终成绩：百分制_____； 五分制_____			
院长签章：		学院盖章：		年 月 日
备 注				

注：1. 毕业设计（论文）成绩由中期检查评分（满分 20 分）、指导教师评分/复议评分（满分 30 分）和答辩小组评分（满分 50 分）相加，得出百分制成绩，再按 100-90 分为“优”、89-80 分为“良”、79-70 分为“中”、69-60 分为“及格”、60 分以下为“不及格”的标准折合成五级分制成绩；

2. 此表原件一式三份，一份存入学生档案，一份装订到毕业论文中，一份学院教务科留存。

一种基于工作量的 Serverless 计算自动伸缩算法的设计与实现

摘 要

Serverless 计算作为一种新型的计算模式，在云计算和边缘计算领域被广泛应用。本研究针对 Serverless 计算的自动伸缩问题，设计了一种主要基于 ARIMA 模型的自动伸缩算法，并对比了其他常见预测模型，对其性能进行了全面的比较评估。该算法结合了负载预测和自动伸缩策略，能够根据应用程序的负载变化动态调整计算资源。并且在基础算法基础上引入了自动选择最优模型超参数的功能，更利于实际应用。通过引入多种预测模型（如 MA、VAR、ARIMA 和 Prophet 模型）并对比和改进，相较于目前主流的基于阈值的伸缩，我们的算法提高了负载预测的准确性，实现了对模型的自动训练、动态调整和优化。

在实现算法部署的过程中，我们在 Kubernetes 集群上设计了一种基于 Prometheus 的监控方案，此方案能够收集 Serverless 应用程序的工作负载数据，动态地根据历史数据，预测下一时间段的负载，进而通过 HPA 控制器自动调整 Serverless 函数的副本数量。

实验结果表明，这种智能化的自动伸缩算法可以有效预测副本数量，提高 Serverless 应用程序的资源利用率，减少计算资源浪费，同时保证应用程序的稳定性和可靠性。

关键词 Serverless 云计算 时序预测 自动伸缩 ARIMA

Design and Implementation of Workload-based Auto-scaling Algorithm for Serverless Computing

ABSTRACT

As a nascent computing paradigm, Serverless computing has gained traction in the domains of cloud and edge computing. This study presents a workload-oriented auto-scaling algorithm designed specifically for Serverless computing. This algorithm, marrying load prediction and auto-scaling policy, can dynamically allocate computational resources in accordance with application load. Leveraging a range of models such as MA, VAR, ARIMA, and Prophet, it seeks to enhance the precision of load prediction. It self-trains and dynamically scales computational resources, continuously optimizing models through data obtained via service monitoring.

When deployed on a Kubernetes cluster, a Prometheus-based monitoring solution is designed in this paper. This solution aggregates metric data from Serverless applications and leverages statistical model training features to forecast future load, subsequently utilizing the HPA controller to auto-adjust the replica count of Serverless functions.

The findings illustrate that the proposed smart algorithm can effectively estimate replica counts, augmenting Serverless application resource utilization, curbing computational resource wastage, and bolstering application stability and reliability.

KEY WORDS Serverless Cloud computing Time series prediction Auto scaling ARIMA

目 录

第一章 引言	1
1.1 背景介绍	1
1.1.1 Serverless 计算概述	1
1.1.2 自动伸缩的重要性与挑战	1
1.2 研究目的与意义	2
第二章 相关工作与理论基础	3
2.1 Serverless 计算的基本原理与特点	3
2.2 Serverless 计算与自动伸缩方法概述	4
2.3 工作量预测方法	4
2.3.1 MA（移动平均，作为基线方法）	5
2.3.2 VAR（向量自回归）	5
2.3.3 ARIMA（自回归差分移动平均）	6
2.3.4 Prophet	7
2.3.4.1 简介	7
2.3.4.2 模型原理	7
2.3.4.3 贝叶斯推断和参数估计	8
第三章 算法的设计与实现	9
3.1 准备工作	9
3.1.1 数据收集与预处理	9
3.1.2 特征工程	10
3.1.3 模型选择与评估	10
3.2 自动伸缩算法设计与实现	13
3.2.1 MA 模型	13
3.2.2 VAR 模型	15
3.2.3 ARIMA 模型	18
3.2.3.1 模型参数选择	18
3.2.3.2 ARIMA 模型的预测效果评估	18
3.2.4 Prophet 模型	23
3.2.4.1 数据预处理	23
3.2.4.2 模型训练	23
3.2.4.3 Prophet 模型的预测效果评估	23

3.2.5	各个模型的预测性能对比	25
3.2.6	各个模型的训练时间对比	25
3.2.7	结论	26
第四章	算法的部署与应用	27
4.1	整体设计	27
4.1.1	算法	27
4.1.2	监控	28
4.1.3	控制	28
4.2	硬件与软件环境	28
4.3	Kubernetes 集群的搭建和 OpenFaaS 的部署	28
4.3.1	OpenFaaS 的部署	28
4.3.1.1	部署函数	29
4.3.1.2	调用函数	30
4.3.1.3	查看部署状态	30
4.4	Prometheus 获取指标	30
4.4.1	根据预测结果调整副本数量	31
4.4.2	自动伸缩: HPA 控制器	32
4.5	流量模拟与伸缩实验	32
4.5.1	请求变化曲线的构造	32
4.5.2	基于模拟流量的自动伸缩仿真	34
4.5.3	仿真结果与结果分析	34
第五章	结论与展望	37
5.1	结论	37
5.1.1	本文的主要成果	37
5.1.2	对实际问题的解决方案	37
5.2	局限性与未来工作	37
5.2.1	本文的不足之处	37
5.2.2	可进一步探讨和优化的方向	38
参考文献		
致 谢		
外 文 资 料		
外 文 译 文		
开 题 报 告		
中 期 检 查 表		
教师指导毕业设计(论文)记录表		

第一章 引言

1.1 背景介绍

1.1.1 Serverless 计算概述

Serverless 计算是云计算的一种新模式，它可以动态地扩展和缩放计算资源，按需自动调配计算资源来运行业务代码。不同于传统的基础设施即服务（IaaS）和平台即服务（PaaS）模式，Serverless 模式管理和操作底层基础设施和服务器，使开发者可以专注于业务逻辑和代码的开发，无需管理服务器和基础架构。

在 Serverless 模式下，资源由事件驱动自动调配，开发者只需编写并部署业务逻辑的代码（往往为小段的功能代码），并为其指定触发事件，云服务商会在事件发生时自动执行对应的代码段。开发者无需提前规划资源或服务器数量，也无需维护服务器及其运营。这大大降低了开发和维护成本，使开发者可以更专注于产品创新。

主流云计算服务商都提供了 Serverless 产品和服务，如 AWS Lambda，Azure Functions，Google Cloud Functions 等。此类产品通过事件驱动执行无服务器代码段，并能够自动适配计算资源，为 Serverless 应用快速扩缩容提供支撑。目前，Serverless 计算正快速发展，在云原生应用、微服务架构以及边缘计算等领域有着广泛的应用。

作为云计算的一种新模式，Serverless 计算具有动态扩展、按需计费、自动运维等显著特点，大大降低了开发和维护成本，为应用的快速开发和创新提供了很好的支撑。其作为云计算发展的方向之一，值得持续关注和研究。

1.1.2 自动伸缩的重要性与挑战

自动伸缩是 Serverless 计算模式下的一项关键功能。它能够动态地按需扩展或缩减计算资源，以适应工作负载的变化，从而提高资源利用率并优化成本。

自动伸缩的重要性在于，在云环境下，工作负载往往是不可预测和突发的。在没有启用自动伸缩时，只能依赖于管理员静态预先配置计算资源来应对业务流量。由于无法准确预测工作负载的模式和峰值，静态预配计算资源很可能导致资源浪费，或无法满足工作负载的需求。动态扩展资源能够在负载增加时迅速提供更多的计算能力，而在需求下降时，也能够快速释放多余的资源，避免资源闲置。这可以显著提高整体资源利用率，并实现按需付费。

自动伸缩也面临一些技术挑战。首先，自动伸缩系统需要准确而高效的负载监控和预测机制，以便能够及时发现负载变化并作出相应调整。其次，资源调度和扩展需要考虑成本优化和服务质量之间的平衡。此外，频繁的资源扩展和释放也会带来性能开销，因此需要进行优化以实现平滑的伸缩。最后，与传统的静态资源池相比，动态变化的资源拓扑也增加了运维的复杂性。

即便面临挑战，自动伸缩已经成为 Serverless 架构的关键。因此，未来工程师需要开发更智能的监控预警手段、更高效的资源调度算法以及更平滑的伸缩机制，以进一步

发挥自动伸缩的优势。这也是 Serverless 计算进一步发展的重要方向之一。

1.2 研究目的与意义

本研究的目的是提出一种适用于 Serverless 计算的自动伸缩算法, 它综合了多种可选算法, 通过负载预测和资源调度策略实现计算资源的动态扩容和缩容, 以满足工作负载的要求, 保证 Serverless 应用的高性能、高可靠。

展开来说, 本研究具有如下具体目标:

1. **实现应用自动伸缩的智能化。** 不同于现有的阈值触发式自动伸缩机制, 本研究通过采用时间序列模型进行短期负载预测, 实现了基于预测的自动伸缩决策, 更加智能和灵活。随着实际负载数据积累, 模型也可以动态优化和调整。
2. **提高应用的资源利用率和运行稳定性。** 通过负载预测机制, 可以更加精确地预测工作负载的变化, 并相应调整计算资源, 避免资源浪费或不足, 保证服务质量。从而可以降低 Serverless 应用的运行成本, 并提高其可靠性。
3. **提供实际平台上的部署和运维的参考实现。** 通过设计一套基于 Prometheus 监控和预警机制, 收集 Serverless 应用的监控指标, 并将数据输入预测模型进行负载预测和自动伸缩决策, 这为 Serverless 应用在 Kubernetes 上的实施提供了参考案例。

本研究希望在实现算法的同时提供可落地的技术方案, 为其进一步推广应用提供了理论和技术支撑。

第二章 相关工作与理论基础

Serverless 计算是一种无服务器的计算模型，它允许开发者构建和运行应用程序而无需关注基础设施。在 Serverless 架构中，第三方服务（如云提供商）负责管理服务器、网络和其他资源。这样，开发者可以专注于编写应用程序逻辑，而无需担心基础设施的维护和扩展。本章节将介绍 Serverless 计算的基本原理和特点，以及相关的研究工作。

2.1 Serverless 计算的基本原理与特点

从用户的角度来看，Serverless 计算是一种弹性、按需付费、无状态的计算模型。

1. **自动弹性伸缩**：Serverless 应用程序可以根据负载自动调整资源，从而实现高效的资源利用率和低延迟。
2. **按需付费**：与预先分配资源的传统计算模型不同，Serverless 计算按实际使用的资源付费，降低了成本。
3. **无状态性**：Serverless 函数通常是无状态的，这意味着它们不会存储任何关于之前请求的信息。这使得 Serverless 函数易于扩展和管理。

而从研发者的角度，Serverless 计算是一种基于事件驱动的、无服务器的计算模型。

在云服务平台实现 Serverless 计算的方式各异。常见的实现载体包括容器化（Containerization）、虚拟化（Virtualization）。例如，AWS Lambda 采用基于 KVM 的虚拟化技术^[1]。

OpenFaaS（Open Function as a Service）是一个开源的 Serverless 计算平台，它允许开发者在 Kubernetes 集群上轻松部署和管理函数。Kubernetes 是一个用于自动部署、扩展和管理容器化应用程序的开源平台。

OpenFaaS 通过以下组件实现 Serverless 计算^[2]：

1. **Gateway**：负责处理请求、调度函数、管理函数生命周期，以及提供 API 和仪表板。
2. **Function Watchdog**：作为每个函数的入口点，用于将传入的 HTTP 请求转发到函数。
3. **Function Pods**：包含函数代码和依赖的容器，它们根据负载自动扩展。

在 Kubernetes 集群上部署 OpenFaaS 具有以下优点：

1. **原生集成**：OpenFaaS 利用 Kubernetes 的功能，如自动扩展、滚动更新和自动恢复。
2. **跨平台支持**：Kubernetes 支持多种云提供商和操作系统，这意味着 OpenFaaS 可以在多个平台上运行。

3. 可观测性和监控: OpenFaaS 可以与 Kubernetes 监控和日志记录工具(如 Prometheus 和 Grafana) 集成

2.2 Serverless 计算与自动伸缩方法概述

自动伸缩是一种在计算资源需求变化时, 动态调整计算资源分配的方法。现有的自动伸缩方法包括基于单一指标伸缩、手动收缩、根据历史定时伸缩和多指标阈值收缩等。

现有这些方法各自有其优点, 但在实际应用中均有局限性。

1. **基于单一指标伸缩**: 该方法根据某个指标, 如 CPU 使用率或内存使用率等, 来调整计算资源。缺点在于, 弹性调整速度较慢, 可能无法适应突发流量, 导致服务性能下降。此外, 如果所选指标不准确或阈值设置不合理, 可能会导致资源浪费或不足。
2. **手动收缩**: 在流量突发时, 手动扩展计算资源以满足需求。缺点在于, 这种方法响应慢, 危险性高, 因为在高峰期可能无法及时扩展资源。此外, 手动收缩无法实现精细化操作, 可能导致资源使用效率低下。
3. **映射历史伸缩**: 通过观察周期性, 复制历史的收缩曲线, 调整计算资源。这种方法无法适应弹性业务需求, 对于实时响应突发流量的能力有限。同时, 无法追踪到历史数据中的趋势性变化, 可能导致资源分配不合理。
4. **多指标阈值收缩**: 在这种方法中, 根据多个指标和预设阈值调整计算资源。尽管这种方法相对于其他方法更为精确, 但其效果取决于指标选取和阈值设定。若未充分考虑这些因素, 可能会影响系统性能, 导致资源分配不合理。并且依旧无法考虑到业务发展的趋势性。

可观察到, 现有的自动伸缩方法均存在不足, 无法在不同场景下完美适应需求。因此, 需要进一步研究和发展更为智能、灵活的自动伸缩策略, 以满足多样化的计算资源需求。

2.3 工作量预测方法

为了保证 Serverless 应用程序在业务中的可靠性, 应当设计一种有效的自动伸缩算法。为此, 我们采用了基于工作量的 Serverless 计算自动伸缩算法, 结合负载预测和自动伸缩策略。在本节中, 我们将详细介绍三种工作量预测方法: 移动平均 (MA)、向量自回归 (VAR) 以及差分整合移动平均自回归 (ARIMA) 模型。

2.3.1 MA（移动平均，作为基线方法）

移动平均（Moving Average，简称 MA）是一种简单的时间序列预测方法，它通过计算一定时间窗口内的平均值来预测未来的数据。在 Serverless 计算的负载预测中，可以采用 MA 方法来预测未来一段时间内的工作量。给定一个时间序列数据集 $\{x_1, x_2, \dots, x_n\}$ ，移动平均法用长度为 m 的滑动窗口来计算序列的平均值，计算公式如下：

$$\hat{x}_{n+1} = \frac{1}{m} \sum_{i=n-m+1}^n x_i \quad \text{式 (2-1)}$$

MA 方法简单易实现，但其预测精度受到时间窗口长度的影响，较长的时间窗口可能导致预测滞后，较短的时间窗口可能导致预测不稳定。

我们选择移动平均作为基准线方法的原因在于，首先它简单易实现：相较于其他复杂的预测方法，移动平均法的实现过程非常简单。只需要计算一定时间窗口内的数据平均值，无需涉及复杂数学模型。这使得移动平均方法易于理解和快速实现。其次，它拥有低计算成本。移动平均法的计算复杂度较低，不需要进行复杂的参数估计和优化。这意味着移动平均法在计算资源有限的场景下，仍然能够保持较好的预测性能。再者，移动平均法具有很好的平滑效果，能够有效消除短期的波动和噪声，使预测结果更加稳定。这对于某些具有高噪声或波动性的场景（如负载预测）来说，是非常有价值的。

2.3.2 VAR（向量自回归）

向量自回归（VAR）模型是一种用于多变量时间序列数据的预测方法。VAR 模型通过将每个时间序列作为其他时间序列的滞后值（lagged values）的线性函数来进行建模。VAR 模型具有预测多元时间序列变量之间关系的优势，可以捕捉变量间的动态相互影响。

假设我们有 n 个时间序列数据 $y_t = (y_{1t}, y_{2t}, \dots, y_{nt})$ ，则 p 阶 VAR 模型可表示为：

$$y_t = c + A_1 y_{t-1} + A_2 y_{t-2} + \dots + A_p y_{t-p} + e_t \quad \text{式 (2-2)}$$

其中， $t = 1, 2, \dots, T$ ， c 是一个 n 维常数向量， A_i 是一个 $n \times n$ 维系数矩阵， e_t 是一个 n 维误差向量，满足 $E(e_t) = 0$ 且协方差矩阵为常数矩阵，即 $E(e_t e_t') = \Sigma$ ， $E(e_t e_{t-j}') = 0$ 对于所有的 $j \neq 0$ 。

举个例子，为了预测 Serverless 计算的工作量，我们首先需要收集多元时间序列数据，例如请求数、响应时间和 CPU 使用率等。接着可以使用 VAR 模型来预测未来的工作量。

假设我们有三个时间序列数据：请求数 x_t 、响应时间 y_t 和 CPU 使用率 z_t ，可以构建一个二阶 VAR 模型如下：

$$\begin{bmatrix} x_t & y_t & z_t \end{bmatrix} = c + A_1 \begin{bmatrix} x_{t-1} & y_{t-1} & z_{t-1} \end{bmatrix} + A_2 \begin{bmatrix} x_{t-2} & y_{t-2} & z_{t-2} \end{bmatrix} + e_t \quad \text{式 (2-3)}$$

通过拟合此 VAR 模型，可以预测未来的工作量，从而为 Serverless 计算提供合适的自动伸缩策略。

2.3.3 ARIMA (自回归差分移动平均)

ARIMA (Autoregressive Integrated Moving Average, 自回归差分移动平均) 是一种广泛应用于时间序列预测的方法。ARIMA 模型由三个主要部分组成: 自回归 (AR), 差分 (I) 和移动平均 (MA)。

给定一个时间序列数据 y_t , ARIMA 模型表示为^[3]:

$$\Phi(B)(1-B)^d y_t = \Theta(B)e_t \quad \text{式 (2-4)}$$

其中, B 是后向移位算子 (即 $By_t = y_{t-1}$), d 是差分阶数, $\Phi(B)$ 和 $\Theta(B)$ 分别是 p 阶自回归多项式和 q 阶移动平均多项式, 形式如下:

$$\Phi(B) = 1 - \phi_1 B - \phi_2 B^2 - \dots - \phi_p B^p \quad \text{式 (2-5)}$$

$$\Theta(B) = 1 - \theta_1 B - \theta_2 B^2 - \dots - \theta_q B^q \quad \text{式 (2-6)}$$

可以将 ARIMA 模型表示为 $\text{ARIMA}(p, d, q)$, 其中 p 是自回归项的阶数, d 是差分的阶数, q 是移动平均项的阶数。

为了预测 Serverless 计算的工作量, 我们首先需要选择合适的 p 、 d 和 q 参数。可以通过观察时间序列的自相关图 (ACF) 和偏自相关图 (PACF) 来确定这些参^[4], 或者使用自动参数选择方法, 如 Akaike Information Criterion (AIC) 或 Bayesian Information Criterion (BIC)。

例如, 假设我们选择了 $\text{ARIMA}(2, 1, 1)$ 模型来预测请求数量 x_t , 模型可以表示为:

$$(1 - \phi_1 B - \phi_2 B^2)(1 - B)x_t = (1 - \theta_1 B)e_t \quad \text{式 (2-7)}$$

通过拟合此 ARIMA 模型, 可以预测未来的工作量, 从而为 Serverless 计算提供合适的自动伸缩策略。

为了提高预测的准确性, 可以结合 MA、VAR 和 ARIMA 模型进行预测和性能对比, 根据实际情况选择合适的方法。

2.3.4 Prophet

2.3.4.1 简介

Prophet 是由 Facebook 的核心数据科学团队开发的开源算法，专门用于进行时间序列预测。它设计的目标是使得时间序列预测对于业务问题更加适用，包括那些存在一些实际的、复杂的趋势的问题，如周期性变化、季节性效应和假日效应等。^[5]

2.3.4.2 模型原理

Prophet 模型基于分解的时间序列预测方法，主要包括三个成分：趋势 (Trend)、季节性 (Seasonality) 和假日效应 (Holidays)。趋势部分旨在捕获数据中的长期趋势，季节性部分则用来捕获周期性模式，而假日效应则针对不规则的事件，例如公众假日或其他特殊事件。

这种分解方法基于以下的加法模型：

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t \quad \text{式 (2-8)}$$

其中， $y(t)$ 是观测值， $g(t)$ 是趋势成分， $s(t)$ 是季节性成分， $h(t)$ 是假日效应， ϵ_t 是误差项。

趋势成分 $g(t)$ ： Prophet 模型假设趋势成分可以通过非线性函数进行适应，而这个非线性函数的形状由数据的变化点决定。在这种设置下，趋势可以在变化点灵活地改变其增长速率。具体来说，对于没有饱和上限和下限的趋势，Prophet 使用以下的分段线性模型：

$$g(t) = \left(k + \sum_{i=1}^S a_i 1(t > \tau_i) \right) t + \left(m + \sum_{i=1}^S b_i 1(t > \tau_i) \right) \quad \text{式 (2-9)}$$

其中， S 是变化点的数量， τ_i 是第 i 个变化点的位置， a_i 和 b_i 是在第 i 个变化点的增长率和偏移量的变化。 k 和 m 是初始化的增长率和偏移量。

季节性成分 $s(t)$ ： Prophet 模型采用傅里叶级数来适应季节性效应，可以捕获不同周期的季节性变化。具体来说，年度和周度季节性模式可以通过以下傅里叶级数进行建模：

$$s(t) = \sum_{n=1}^N \left(a_n \cos \left(\frac{2\pi nt}{P} \right) + b_n \sin \left(\frac{2\pi nt}{P} \right) \right) \quad \text{式 (2-10)}$$

其中， P 是季节性周期， N 是傅里叶级数的阶数， a_n 和 b_n 是要估计的参数。

假日效应 $h(t)$ ：

在许多实际应用中，特定的日期或事件（如公众假期，特定的营销活动或其他特殊事件）可能会对时间序列产生显著影响。这些影响可以被视为“假日效应”，它们会在特定日期或事件期间突然改变时间序列的行为。

这种假日效应可以通过添加额外的回归项来建模。具体来说,对于每一个假日,都会创建一个指示函数,该函数在假日期间为 1,其他时间为 0。然后,这个指示函数与一个回归系数相乘,该系数表示假日的影响强度。所有的假日效应可以表示为:

$$h(t) = \sum_{i=1}^H k_i 1(t \in D_i) \quad \text{式 (2-11)}$$

其中, H 是假日的数量, D_i 是第 i 个假日的日期, k_i 是对应的回归系数, $1(\cdot)$ 是指示函数。

假日效应的参数 k_i 是通过模型拟合过程中的数据来估计的。由于假日效应可能在不同的假日之间有所不同,每个假日都有自己的回归系数。

在实际应用中,用户需要提供一个假日的列表,并指定每个假日的日期。此外,用户还可以为每个假日设置一个“窗口”,表示假日效应可以影响的日期范围。

2.3.4.3 贝叶斯推断和参数估计

Prophet 模型对模型参数的估计是基于贝叶斯推断的。这意味着我们不仅仅是找到一个最优的参数值,而是要找到参数的概率分布,这样我们就可以考虑到参数估计的不确定性。

贝叶斯推断的基本思想是结合先验知识和观测数据来估计模型参数。对于每一个参数,应当指定一个先验分布,这个分布表示了观测数据之前我们对参数的信念。然后,我们使用观测数据来更新这个信念,得到参数的后验分布。

在 Prophet 模型中,我们对每一个参数 θ (可以是趋势、季节性或假日效应的参数) 的先验分布 $p(\theta)$ 做出某些假设。然后,我们使用观测数据 y 来计算参数的似然函数 $L(\theta|y)$, 这个函数表示在给定参数 θ 的情况下,观测数据 y 的概率。最后,我们使用贝叶斯定理来计算参数的后验分布:

$$p(\theta|y) = \frac{L(\theta|y)p(\theta)}{p(y)} \quad \text{式 (2-12)}$$

在这个公式中, $p(y)$ 是观测数据的概率,我们通常不直接计算这个值,而是通过对参数的所有可能值的后验分布进行归一化来得到。

在实际的计算中,由于后验分布可能是复杂的,我们通常使用马尔科夫链蒙特卡洛 (MCMC) 方法^[6] 或者变分贝叶斯方法^[7] 来近似计算后验分布。

此处,参数的先验分布通常是正态分布或者拉普拉斯分布,这取决于我们对参数的先验知识。例如,如果我们认为一个参数应该接近 0,那么可以选择均值为 0 的正态分布作为先验分布。

第三章 算法的设计与实现

算法的设计和实现，需要基于对时间序列数据的特性和规律的分析 and 建模。这些特性包括趋势、季节性、周期性、自相关性等。基于这些特性，可以设计出各种不同的时间序列预测算法。前文已经介绍了本研究将采用的预测算法，在实际使用时间序列预测算法进行预测之前，原始的数据可能存在缺失值和异常值、数据的格式不规范等问题。应当对数据进行预处理。在此之后基于标准化后的数据应用具体的算法。整体设计思路如图 3-1 所示。

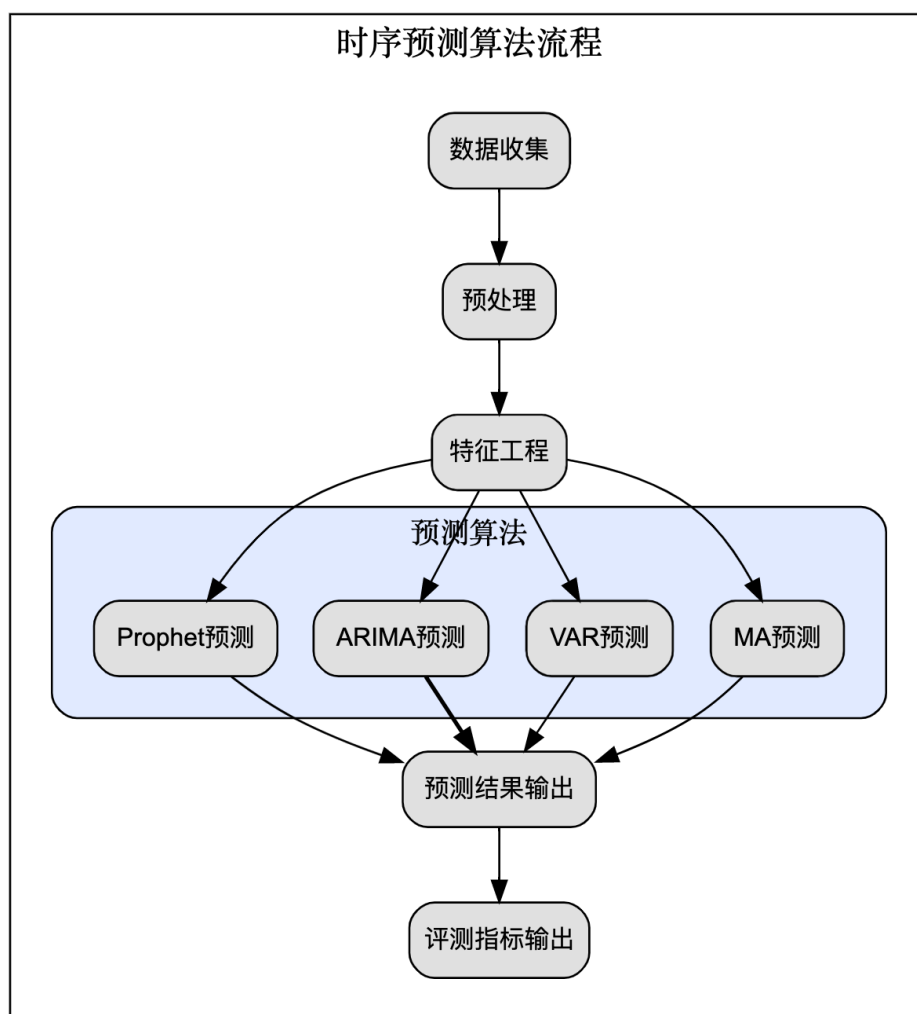


图 3-1 时序算法预测流程

3.1 准备工作

3.1.1 数据收集与预处理

数据收集和预处理是时间序列预测任务的关键步骤，可以确保我们拥有无缺失值、无异常值、可靠和有代表性的数据，以便进行准确的预测。在本研究中，我们使用 **Binance**

数字货币交易所交易 API 调用历史数据公开数据集作为数据来源。

我们从 Binance 提供的公开数据集中获取交易 API 调用历史数据。原始数据以 zip 格式存储，每日一个文件。对于每个文件，解压后得到一个 csv 文件，在这个 csv 文件中，每行代表一次交易调用记录。数据包含多个字段，其中 time 字段表示调用发生的时间，单位为毫秒时间戳。其余字段本研究不涉及。

为便于分析和预测，应当对原始数据进行预处理。首先，将毫秒时间戳转换为秒时间戳，以便于后续处理。接着，对数据进行聚合处理，将每个小时内的调用次数汇总，生成一个新的数据集。新数据集包含两列：time 和 event_count。其中，time 代表小时所在的秒时间戳，event_count 代表该小时时间段内的调用次数。

预处理的具体步骤如下：

1. 加载原始数据集。
2. 按小时对数据进行分组聚合，计算每个小时内的调用次数。
3. 将 time 字段的毫秒时间戳转换为秒时间戳。
4. 生成新的数据文件，每个文件是一天的调用统计，包含 time 和 event_count 两列。
5. 将每个月的所有文件聚合为一个文件

经过上述预处理步骤，我们得到了一个无缺失值、无异常值、有序的数据集。

3.1.2 特征工程

特征工程是机器学习和预测任务中的重要环节，可以提取有意义的信息以提高预测性能。由于调用频次与时间通常有较强的相关性，我们从时间戳中提取 weekday 和 hour 两列特征。

图 3-2 展示不同时间点的调用次数分布，可直观观察到数据并非噪声或随机游走，而是存在一定的规律性，因此存在被预测的可能。图 3-3 展示一天中不同小时的调用次数分布，可直观观察到存在峰值时段和谷值时段。图 3-4 a 展示一周中不同天（周一至周日）的调用次数分布。图 3-4 b 展示 time、event_count、weekday 和 hour 之间的相关性矩阵。

通过分析上述图表和相关性矩阵，可以发现调用次数与 weekday 和 hour 特征具有一定的相关性。这些特征将有助于我们建立更准确的工作量预测模型。

3.1.3 模型选择与评估

在本文中，我们采用了 MA、VAR、ARIMA、Prophet 四种时间序列模型来进行负载预测。相较于基于神经网络的机器学习模型，我们选择此类统计学模型的原因主要有以下方面：

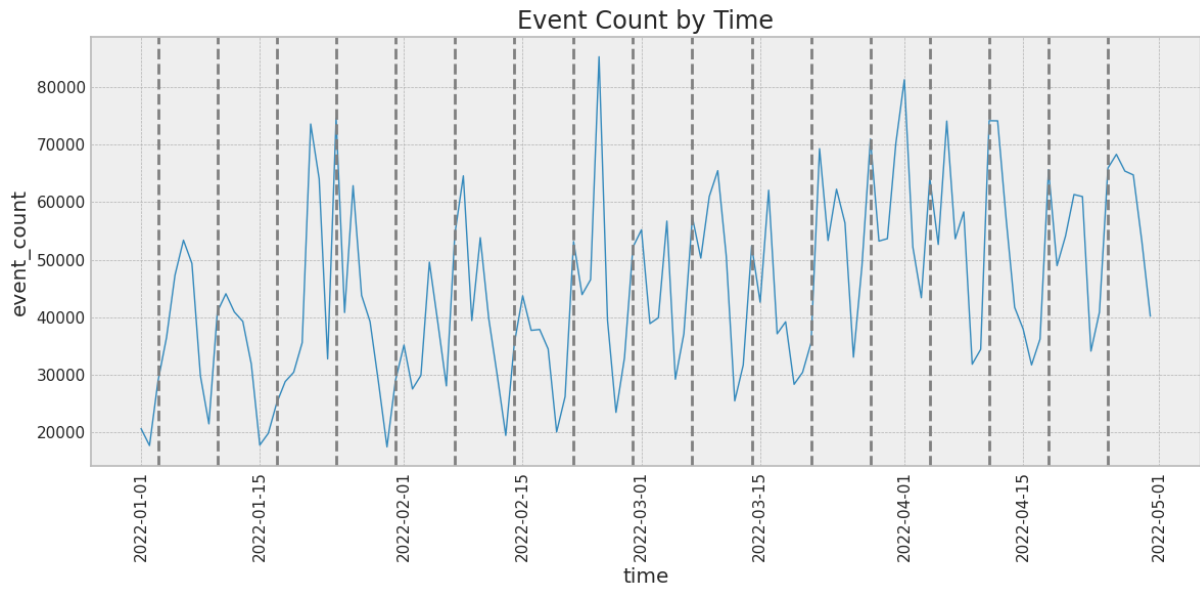


图 3-2 事件计数随时间变化图

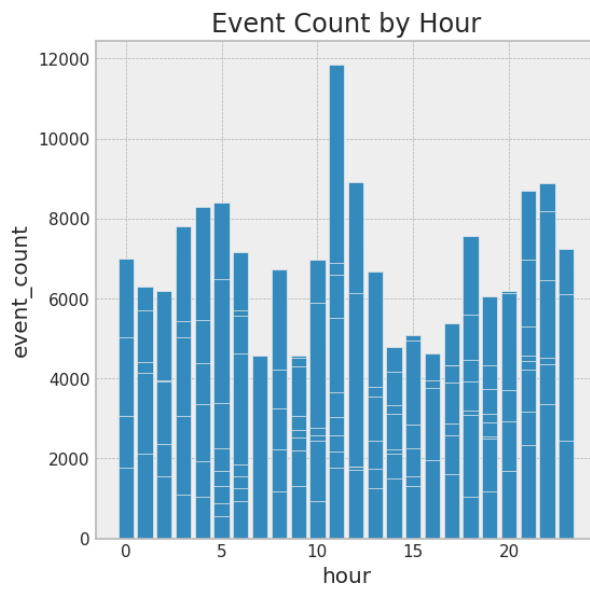
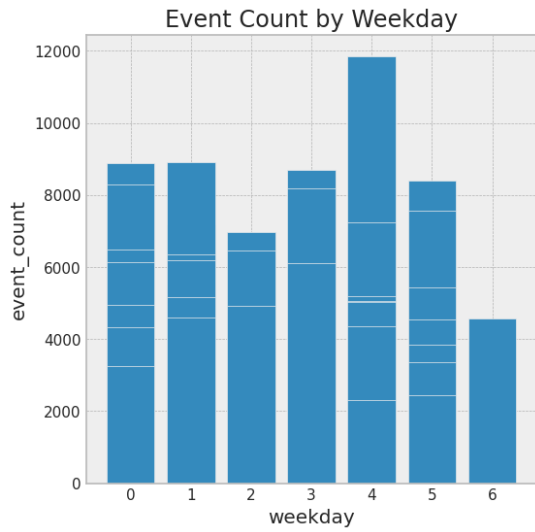


图 3-3 事件计数分布（按小时）

(a) 事件计数分布图 (按 weekday)



(b) 相关性矩阵图

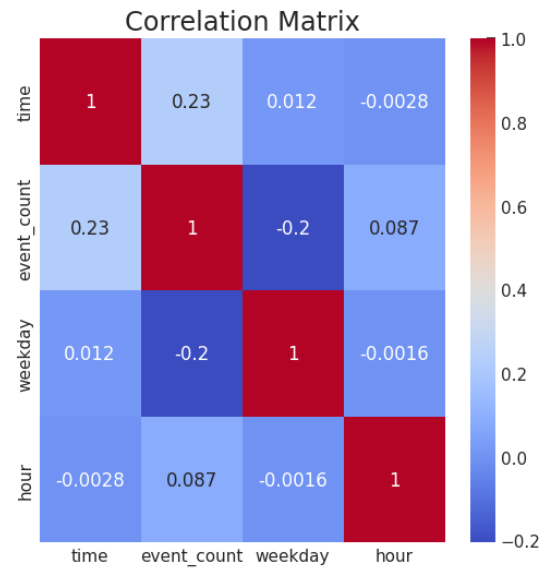


图 3-4 weekday 分布和相关性矩阵

表 3-1 自相关和偏自相关在不同滞后阶数的值

滞后阶数 (Lags)	自相关性 (Autocorrelation)	偏自相关性 (PACF)
0	1.00	1.00
1	0.60	0.60
2	0.44	0.13
3	0.34	0.06
4	0.29	0.06
5	0.27	0.07
6	0.24	0.03
7	0.19	-0.02
8	0.16	0.01
9	0.16	0.04
10	0.18	0.06
11	0.16	-0.00
12	0.16	0.04
13	0.16	0.03
14	0.12	-0.03
15	0.11	-0.00
16	0.11	0.03
17	0.13	0.04
18	0.17	0.07
19	0.18	0.04
20	0.17	0.01
21	0.18	0.04
22	0.21	0.08
23	0.24	0.07
24	0.27	0.07

第一，时间序列模型适用于具有时间依赖性的数据，可以更好地捕捉数据的周期性、趋势性和季节性等特征。在 Serverless 计算中，负载数据通常具有时间相关性，因此时间序列模型是一种自然的选择。

第二，时间序列模型具有可解释性和可靠性，可以更好地理解模型的运作原理，评估模型的准确性和稳定性。对于负载预测来说，准确性和稳定性是至关重要的，因此时间序列模型可以为我们提供更可靠的负载预测结果。

最后，非常重要的一点是，时间序列模型相较于机器学习模型，需要的数据量更小，模型训练和预测速度更快。在 Serverless 计算中，资源的快速响应是至关重要的，而历史数据量往往只有几个星期，使用机器学习模型难以较好收敛，因此时间序列模型可以更快地生成负载预测结果，以满足 Serverless 应用程序对计算资源的实时需求。

本研究选择的各模型各有特点。MA (Moving Average) 模型主要用于捕捉负载数据的短期波动性，本研究选择其作为基准 (Baseline) 模型。VAR (Vector Autoregression) 模型则可以同时考虑多个变量之间的相互作用，从而验证时间中提取的特征是否能改善预测效果，ARIMA (Autoregressive Integrated Moving Average) 模型则是一种广泛应用的时间序列模型，可以同时考虑趋势、周期和季节性等因素，并且能够自动从数据中学习到这些因素，相比 VAR 在保证训练效率的前提下，更加智能。Prophet 则是一个更加新颖的模型，能够考虑到节假日等因素的影响。通过多种模型的组合和比较，从而在实际应用选择最适合负载预测的模型。

3.2 自动伸缩算法设计与实现

3.2.1 MA 模型

MA (Moving Average)，即移动平均模型是一种常用的时间序列模型，用于捕捉负载数据的短期波动性。在本文中，本研究将 MA 模型作为基准模型 (Baseline Model) 来进行负载预测，以评估其他模型的优劣性。

具体地，本研究采用了 WIN_SIZE=3 的 MA 模型来进行负载预测。将历史数据按照 WIN_SIZE 划分成若干个时间窗口，每个时间窗口内的数据点的平均值被用来作为该时间窗口的预测值。通过这样的方式，可以捕捉负载数据的短期波动性，同时减少数据的随机波动性。

为了评估 MA 模型（以及后续其他模型）的预测性能，本研究使用均方误差 (Mean Squared Error)、均方根误差 (Root Mean Squared Error)、平均绝对误差 (Mean Absolute Error)、平均绝对百分比误差 (Mean Absolute Percentage Error) 和对称平均绝对百分比误差 (Symmetric Mean Absolute Percentage Error) 等指标来评估模型的预测准确性和稳定性。在 WIN_SIZE=3 的情况下，我们得到了 3-2 所示模型预测性能。

图 3-6 展示 MA 模型的预测结果和实际负载值的对比图。从图中可以看出，MA 模型能够较为准确地捕捉到负载数据的短期波动性，预测结果与实际负载值的变化趋势基本一致，但预测效果不是很理想。同时，此图也包含 MA 模型的残差图，可观察到 MA

表 3-2 MA 模型评测指标

指标	数值
MSE	1139965.31
RMSE	1067.69
MAE	805.27
MAPE	39.11%
SMAPE	34.47%

模型的残差依旧存在一些周期性，这表明存在改进的空间。

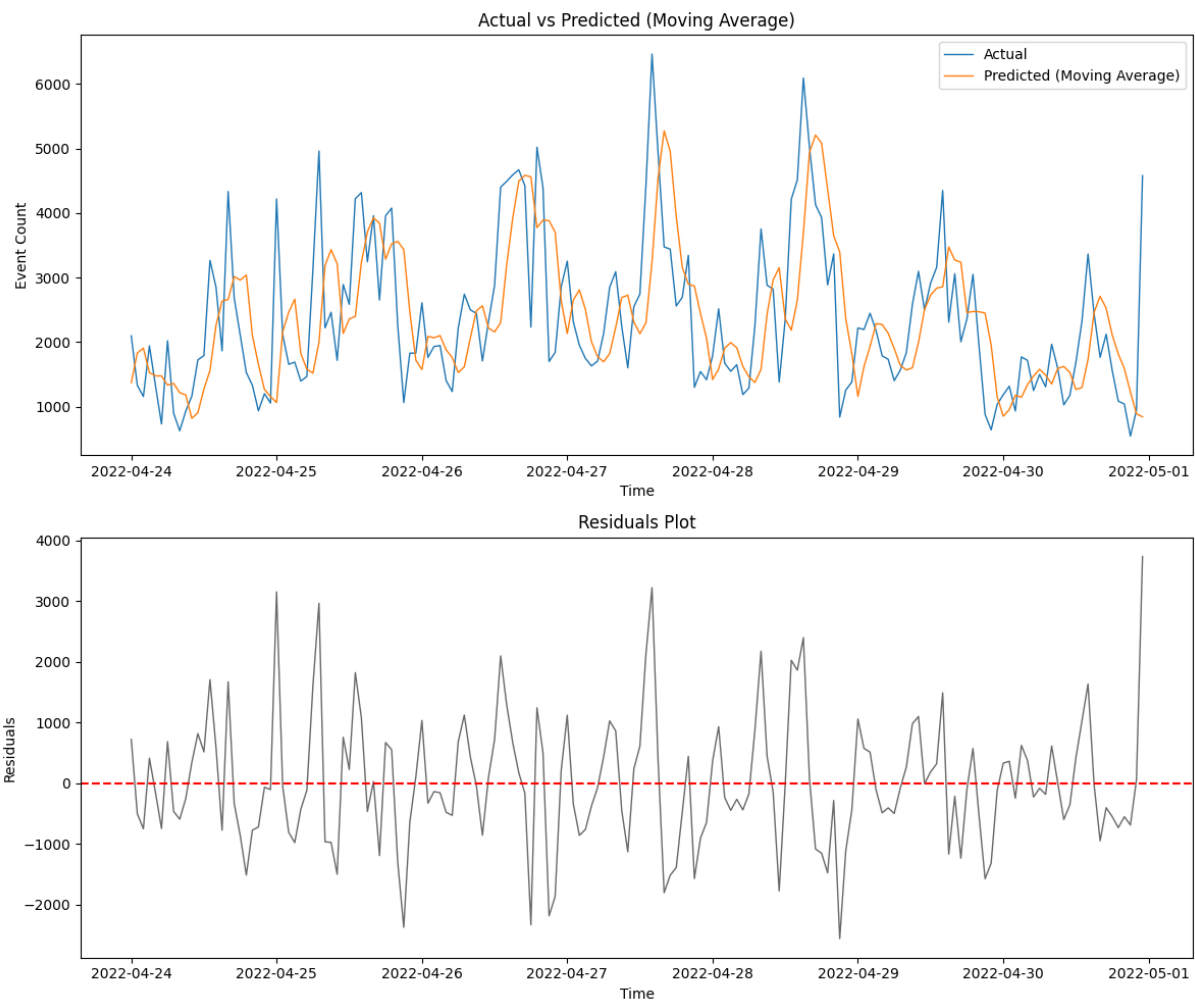


图 3-5 MA 预测结果和实际负载值、残差对比

MA 模型可以为我们提供一个基础的负载预测结果，同时也可以用来评估其他更复杂的负载预测模型的性能优劣。作为基准模型，本研究不期望其实际应用效果，原因在于 MA 模型虽然简单，但在某些情况下可能不适用。例如，在负载数据存在长期趋势或季节性变化的情况下，MA 模型可能无法捕捉到这些特征，从而导致负载预测的准确性下降。

表 3-3 VAR 不同阶数下的各准则值

VAR Order	AIC	BIC	FPE	HQIC
0*	19.34	19.35	2.518e+08	19.35
1	15.22	15.25	4.090e+06	15.23
2	15.18	15.23	3.907e+06	15.20
3	15.17	15.23	3.858e+06	15.19
4	15.16	15.25	3.823e+06	15.19
5	15.15	15.26	3.781e+06	15.19
6	15.14	15.27	3.748e+06	15.18
7	15.13	15.28	3.725e+06	15.19
8	15.12	15.29	3.693e+06	15.18
9	15.10	15.29	3.615e+06	15.17
10	15.08	15.29	3.546e+06	15.16
11	15.07	15.30	3.497e+06	15.15
12	15.05	15.30	3.421e+06	15.14
13	15.02	15.30	3.342e+06	15.12
14	14.99	15.29	3.251e+06	15.10
15	14.96	15.28	3.144e+06	15.08
16	14.91	15.25	2.994e+06	15.03
17	14.85	15.21	2.826e+06	14.98
18	14.77	15.15	2.599e+06	14.91
19	14.66	15.06	2.322e+06	14.80
20	14.48	14.90	1.942e+06	14.63
21	14.20	14.64	1.473e+06	14.36
22	13.62	14.08	8.196e+05	13.78
23	-41.31	-40.82	1.152e-18	-41.13
24	-46.04	-45.53	1.016e-20	-45.85

3.2.2 VAR 模型

VAR 模型是一种常用的多变量时间序列模型，可以同时考虑多个变量之间的相互作用，适用于具有相互关联的多个时间序列数据的预测。在本文中，我们选用 `time`、`weekday` 和 `hour`、`event_count` 等多个特征来构建 VAR 模型。

为了解决模型选择问题，评估模型的好坏，并寻找最佳的模型我们使用一些准则，包括赤池信息准则（Akaike Information Criterion, AIC）、贝叶斯信息准则（Bayesian Information Criterion, BIC）、预测误差准则（Final Prediction Error, FPE）以及汉纳-奎因准则（Hannan-Quinn Information Criterion, HQIC）。

赤池信息准则（Akaike Information Criterion, AIC）：AIC 是由 Akaike 提出的^[8]，主要用于模型选择。AIC 不仅考虑了模型的拟合度，还对模型的复杂度进行了惩罚。当模型越复杂（参数越多）时，AIC 值就越大。选择最佳模型时，通常选择 AIC 值最小的模型。AIC 的计算公式如下：

$$AIC = 2k - 2\ln(L)$$

表 3-4 VAR 模型评测指标

指标	数值	MA 模型基准
MSE	680996.57	1139965.31
RMSE	825.23	1067.69
MAE	590.93	805.27
MAPE	26.98%	39.11%
SMAPE	25.54%	34.47%

其中, k 是模型参数的数量, L 是模型的最大似然值。

贝叶斯信息准则 (Bayesian Information Criterion, BIC) : BIC 也是用于模型选择的一种准则, 由 Schwarz 提出^[9]。与 AIC 类似, BIC 也对模型的复杂度进行了惩罚, 但是 BIC 的惩罚项比 AIC 更大, 因此 BIC 对模型的复杂度惩罚更严重。BIC 的计算公式如下:

$$BIC = \ln(n)k - 2 \ln(L)$$

其中, n 是观察的数据量, k 是模型参数的数量, L 是模型的最大似然值。

预测误差准则 (Final Prediction Error, FPE) : FPE 由 Akaike 提出, 主要用于时间序列模型的选择。FPE 不仅考虑了模型的拟合度, 还对模型的复杂度进行了惩罚^[10]。FPE 的计算公式如下:

$$FPE = \frac{(n+k)p}{n-p}$$

其中, n 是观察的数据量, p 是模型参数的数量。

汉纳-奎因准则 (Hannan-Quinn Information Criterion, HQIC) : HQIC 由汉纳和奎因提出, 主要用于模型选择。与 AIC 和 BIC 类似, HQIC 也对模型的复杂度进行了惩罚, 但是 HQIC 的惩罚项介于 AIC 和 BIC 之间^[11]。HQIC 的计算公式如下:

$$HQIC = -2 \ln(L) + 2k \ln(\ln(n))$$

其中, n 是观察的数据量, k 是模型参数的数量, L 是模型的最大似然值。

表 3-3 展示不同阶数的 VAR 模型的 AIC、BIC、FPE 和 HQIC 值。从表中可观察到, 当阶数为 22 时, VAR 模型的 AIC、BIC、FPE 和 HQIC 值均达到最小, 并且没有变成负值, 因此我们选择阶数为 22 的 VAR 模型。

表 3-4 展示使用 VAR 模型进行负载预测的评价指标。

实验结果表明, 使用 VAR 模型进行负载预测的效果明显优于 MA 模型。

图 3-6 展示使用 VAR 模型进行负载预测的实验结果。可观察到预测值与实际值的误差较小, 且残差图表明基本消除了周期性, 且预测误差更加均匀, 这说明 VAR 模型

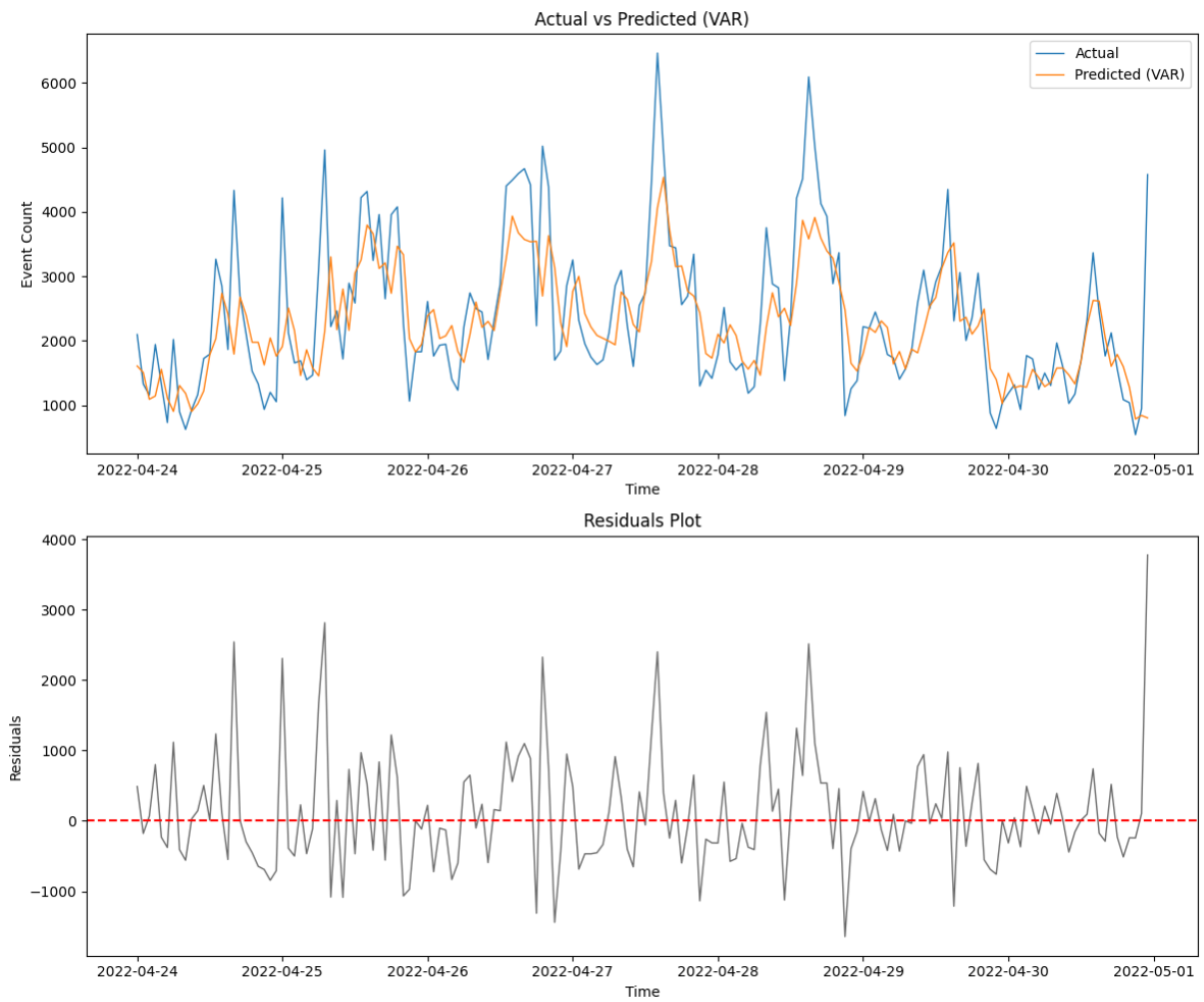


图 3-6 预测结果和实际负载值、残差对比

在捕捉负载数据的周期性、趋势性和季节性等方面具有较好的能力。

3.2.3 ARIMA 模型

3.2.3.1 模型参数选择

在时间序列分析中, ARIMA (Autoregressive Integrated Moving Average) 模型是一种广泛应用的时间序列预测方法。ARIMA 模型的参数选择是关键步骤之一, 本节将介绍基于信息准则的模型参数选择方法。

首先, 应当对时间序列数据进行平稳性检验, 若序列不满足平稳性, 则需要差分处理, 直至达到平稳状态。通过对差分后的序列进行自相关函数 (ACF) 和偏自相关函数 (PACF) 的分析, 可以初步确定 ARIMA 模型的参数 p 和 q 。其中, p 表示 AR 模型的阶数, q 表示 MA 模型的阶数。

在本研究中, ARIMA 模型的参数选择过程采用了 Akaike Information Criterion (AIC) 和 Bayesian Information Criterion (BIC) 两个指标。AIC 和 BIC 均可以用于衡量模型拟合优度, 它们分别对应模型拟合优度与模型复杂度的平衡, 较小的 AIC 或 BIC 值表示更好的模型拟合。

首先, 我们根据给定数据, 计算后生成自相关函数 (ACF) 和偏自相关函数 (PACF) 图表, 如表 3-5 所示。

然后, 我们根据 ACF 和 PACF 图形 (如图 3-7 所示) 确定 ARIMA 模型的参数 p 和 q 。从图中可观察到, ACF 图在滞后阶数为 1 时截尾, PACF 图在滞后阶数为 2 时截尾, 因此可以初步确定 ARIMA 模型的参数 $p=1$, $q=2$ 。

为了进一步确定 ARIMA 模型的参数, 选取不同的 ARIMA 模型参数进行尝试。采用的是穷举并对比 AIC 和 BIC 的方法。

表 3-6 展示针对给定时间序列数据的不同 ARIMA 模型的 AIC 和 BIC 值。

经过比较, 我们发现 ARIMA(2, 0, 2) 模型具有最低的 AIC 值为 47672.61, 同时 BIC 值为 47708.40, 因此选取 ARIMA(2, 0, 2) 作为最优模型。

3.2.3.2 ARIMA 模型的预测效果评估

图 3-8 展示 ARIMA 模型的预测结果和实际负载值的对比图。从图中可以看出, ARIMA 模型相比 MA 和 VAR 模型, 能够更为准确地捕捉到负载数据的短期波动性, 预测结果与实际负载值的变化趋势基本一致, 预测效果较好。同时, 图中还包含 ARIMA 模型的残差图, 可观察到周期性基本被消除。

从表 3-7 中可观察到, 相较于 MA 和 VAR 模型, ARIMA 模型的预测效果评估指标表现良好。均方误差和均方根误差较小, 说明模型的预测误差较小; 平均绝对误差和平均绝对百分比误差也较小, 说明模型的平均预测误差不大; 对称平均绝对百分比误差也较小, 说明模型对正负误差的惩罚是比较对称的。

表 3-5 ACF 和 PACF 表

lags	autocorrelation	pacf
0	1.00	1.00
1	0.60	0.60
2	0.44	0.13
3	0.34	0.06
4	0.29	0.06
5	0.27	0.07
6	0.24	0.03
7	0.19	-0.02
8	0.16	0.01
9	0.16	0.04
10	0.18	0.06
11	0.16	-0.00
12	0.16	0.04
13	0.16	0.03
14	0.12	-0.03
15	0.11	-0.00
16	0.11	0.03
17	0.13	0.04
18	0.17	0.07
19	0.18	0.04
20	0.17	0.01
21	0.18	0.04
22	0.21	0.08
23	0.24	0.07
24	0.27	0.07

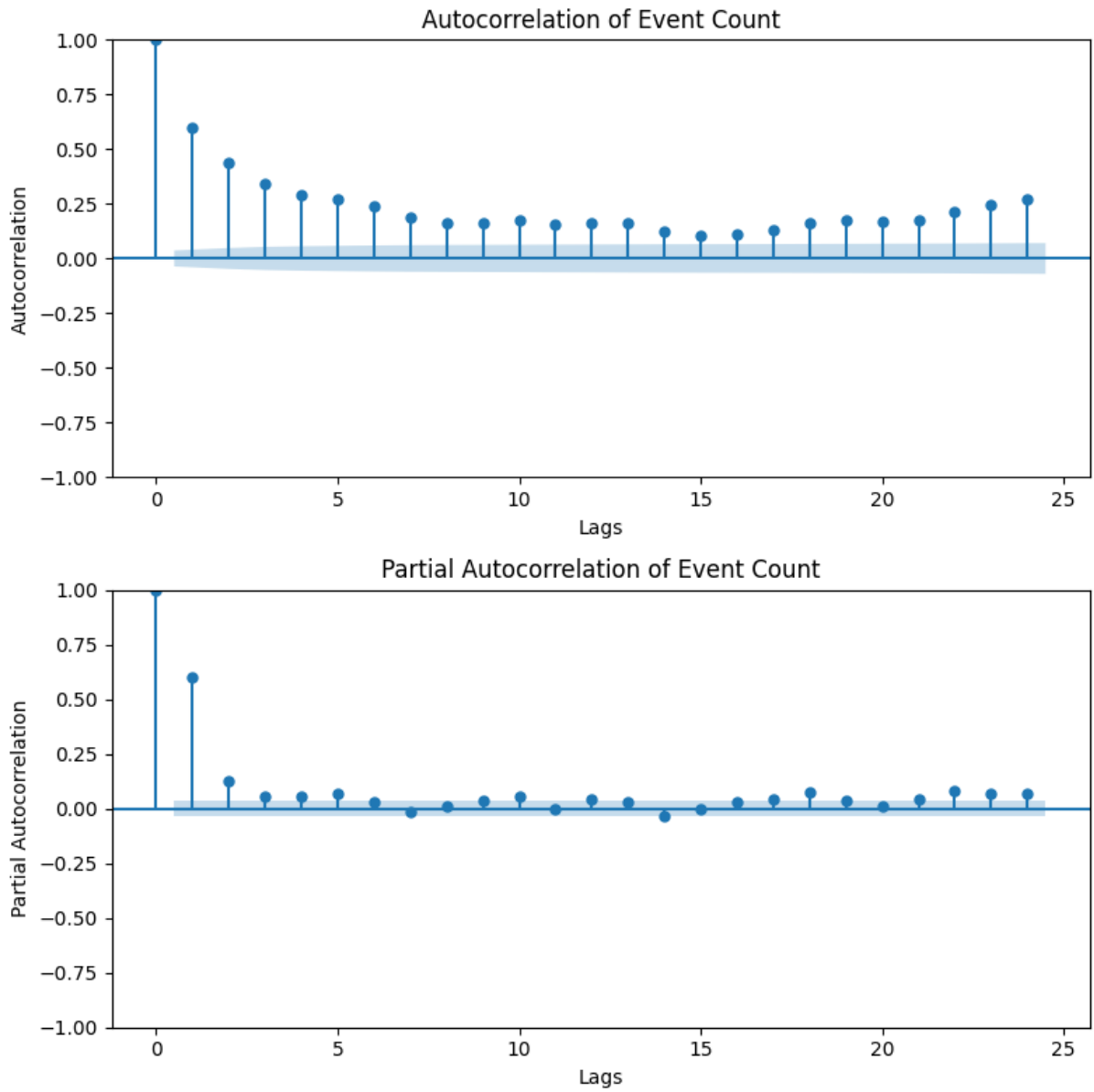


图 3-7 ACF 和 PACF 图形

表 3-6 ARIMA 模型评测指标

ARIMA Parameters	AIC	BIC
(0, 0, 0)	49040.07	49052.00
(0, 0, 1)	48195.20	48213.10
(0, 0, 2)	47930.20	47954.06
(0, 0, 3)	47843.45	47873.28
(0, 0, 4)	47811.97	47847.77
(0, 0, 5)	47786.11	47827.87
(0, 0, 6)	47739.97	47787.69
(0, 0, 7)	47716.49	47770.18
(1, 0, 0)	47764.59	47782.48
(1, 0, 1)	47707.93	47731.79
(1, 0, 2)	47697.82	47727.65
(1, 0, 3)	47687.50	47723.29
(1, 0, 4)	47680.53	47722.29
(1, 0, 5)	47681.68	47729.40
(1, 0, 6)	47681.96	47735.65
(1, 0, 7)	47671.27	47730.93
(2, 0, 0)	47718.77	47742.63
(2, 0, 1)	47679.91	47709.74
(2, 0, 2)	47672.61	47708.40
(2, 0, 3)	47674.64	47716.40
(2, 0, 4)	47675.70	47723.42
(2, 0, 5)	47683.87	47737.56
(2, 0, 6)	47685.69	47745.35
(2, 0, 7)	47671.74	47737.36

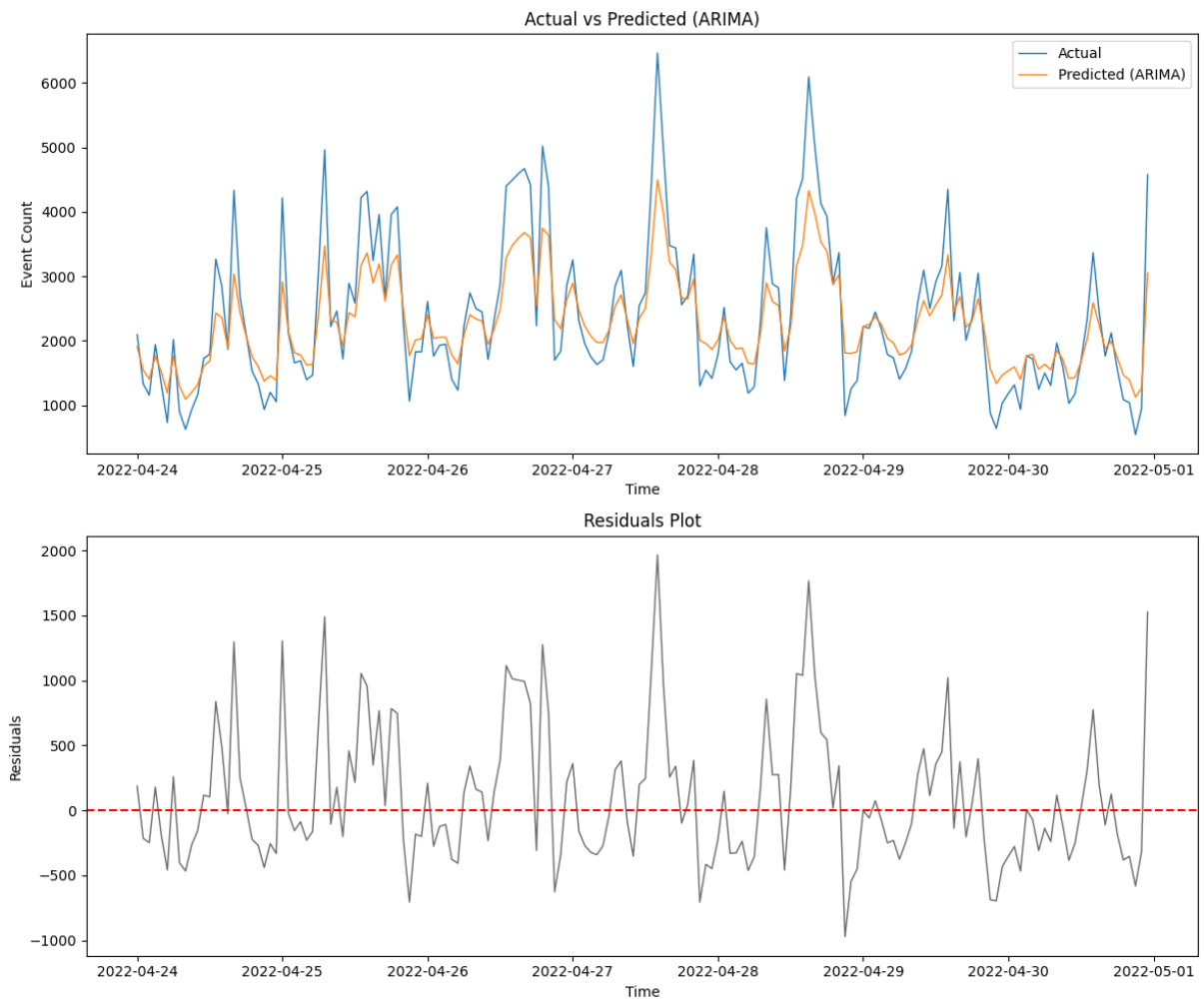


图 3-8 预测结果和实际负载值、残差对比

表 3-7 ARIMA 模型预测效果指标

指标	数值	MA 模型基准
MSE	291700.66	1139965.31
RMSE	540.09	1067.69
MAE	405.24	805.27
MAPE	19.30%	39.11%
SMAPE	17.82%	34.47%

3.2.4 Prophet 模型

3.2.4.1 数据预处理

由于 Prophet 模型所需要的数据格式特殊，首先需要对数据再次预处理。

Prophet 模型要求时间序列的时间戳是 `ds`（日期戳）列，并且它必须是 `datetime` 类型。在本例中，时间戳数据被存储在 `time` 列中，并且以秒为单位。故使用 `pd.to_datetime` 函数将其转换为 `datetime` 类型。

然后，Prophet 模型要求时间序列的值被存储在 `y` 列中。在本研究的数据中，这些值被存储在 `event_count` 列中，因此应当将列名更改为 `y`。

最后，应当将数据集划分为训练集和测试集。在这个例子中，我们选择了 2022 年 1 月 1 日到 2022 年 4 月 30 日的数据，并且将最后一周的数据作为测试集，其余的数据作为训练集。

3.2.4.2 模型训练

在完成数据预处理后，可以开始训练 Prophet 模型。模型训练主要包括模型的初始化和拟合。

首先，应当创建一个 Prophet 模型的实例。在默认情况下 Prophet 模型会自动处理趋势和季节性，但是也可以通过设置参数来调整模型的行为。例如，可以通过设置 `seasonality_mode` 参数来改变季节性的模型（加性或乘性），或者通过设置 `changepoint_prior_scale` 参数来改变趋势变化点的灵敏度。在本例中，我们使用默认设置创建一个 Prophet 模型的实例。

3.2.4.3 Prophet 模型的预测效果评估

图 3-9 展示 Prophet 模型的预测效果。蓝色曲线代表实际值，黄色曲线代表预测值的期望。我们还可以看到一个灰色区域，灰色区域的上边界代表预测值的上界，下边界代表预测值的下界。

图 3-10 展示 Prophet 模型的预测残差图。

我们对 Prophet 模型的预测性能进行了定量评估，所得结果如表 3-8 所示。结果表明，Prophet 模型在所有指标上均优于 MA 模型基准。

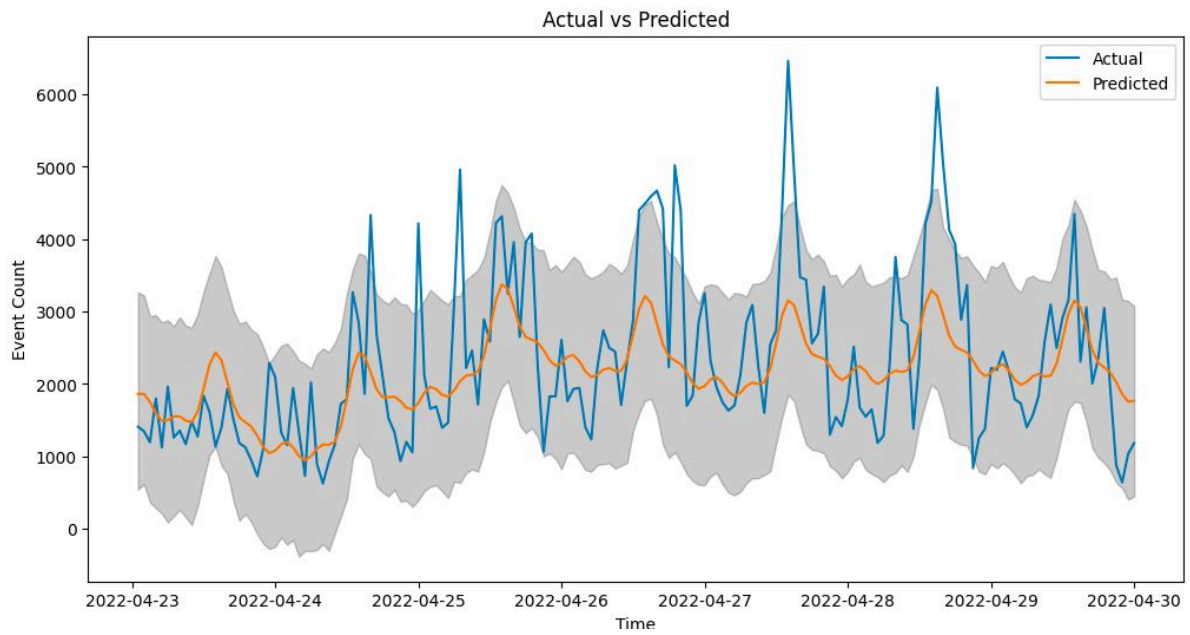


图 3-9 预测结果和实际负载值对比

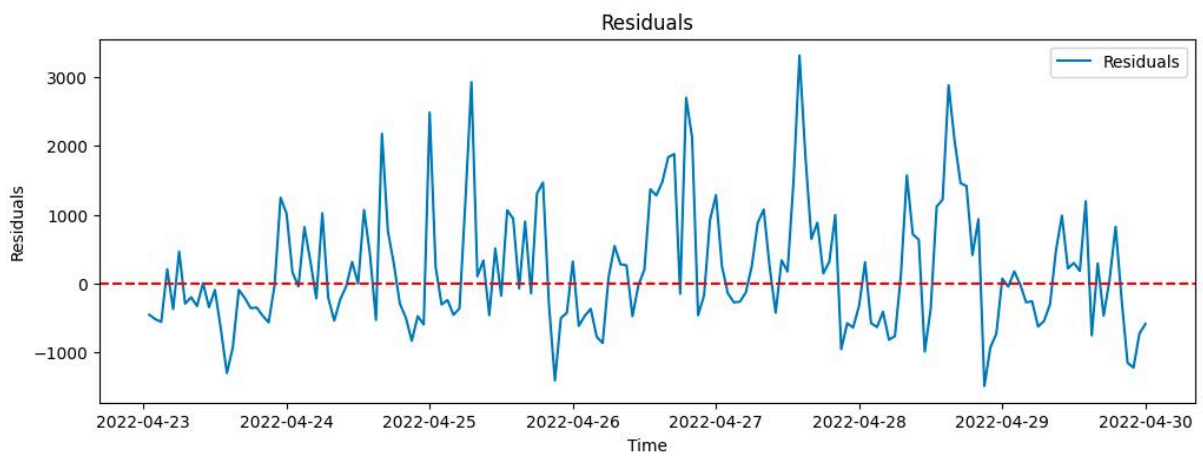


图 3-10 预测结果和实际负载值的残差对比

表 3-8 Prophet 模型预测效果指标

指标	数值	MA 模型基准
MSE	823243.67	1139965.31
RMSE	907.33	1067.69
MAE	673.04	805.27
MAPE	31.22%	39.11%
SMAPE	29.23%	34.47%

3.2.5 各个模型的预测性能对比

表 3-9 模型预测效果对比

指标	MA 模型	VAR 模型	ARIMA 模型	Prophet 模型
MSE	1139965.31	680996.57	291700.66	823243.67
RMSE	1067.69	825.23	540.09	907.33
MAE	805.27	590.93	405.24	673.04
MAPE	39.11%	26.98%	19.30%	31.22%
SMAPE	34.47%	25.54%	17.82%	29.23%

从表 3-9 中我们可观察到,四种模型的预测效果在各个指标上有所不同。ARIMA 模型在所有评测指标上的表现都优于其他三种模型。ARIMA 模型的 MSE、RMSE、MAE、MAPE 和 SMAPE 分别为 291700.66、540.09、405.24、19.30% 和 17.82%,这些指标的数值均低于其他模型,显示出 ARIMA 模型具有更高的预测精度。

VAR 模型的 MSE、RMSE、MAE、MAPE 和 SMAPE 数值为 680996.57、825.23、590.93、26.98% 和 25.54%,排名次之,也显示出良好的预测性能。然后是 Prophet 模型,虽然其在所有指标上的表现都优于 MA 模型,但是与 ARIMA 和 VAR 模型相比则略显逊色。

最后,MA 模型的预测效果最差,所有评测指标的数值都最高,显示出预测误差较大。

综上,从预测精度角度来看,在本任务中,ARIMA 模型是四种模型中表现最好的,然后是 VAR 模型,Prophet 模型和 MA 模型的预测效果相对较差。

3.2.6 各个模型的训练时间对比

表 3-10 模型训练时间对比

模型	训练时间 (s)
MA 模型	0.0
VAR 模型	0.4
ARIMA 模型	0.1
Prophet 模型	4.7

从表 3-10 中我们可观察到,四种模型的训练时间存在显著差异。其中,MA 模型的训练时间最短,几乎为零,表示该模型可以即时进行预测。其次是 ARIMA 模型,训练

时间为 0.1s，也是非常快。VAR 模型的训练时间稍长，为 0.4s。

然而，Prophet 模型的训练时间最长，达到了 4.7s。这可能是由于 Prophet 模型在构建过程中包含了更多的复杂度和参数，因此需要更长的训练时间。这可能在实时或者需要快速响应的预测任务中会成为一种限制。

因此，在选择模型时，除了考虑预测精度外，我们还需要根据具体应用场景和需求，考虑模型的训练时间，以达到最佳的性能和效率的平衡。在本研究的场景中，Serverless 流量预测需要滚动训练模型，因此如果涉及的服务较多，历史数据较长，应当在应用时增加对训练时间的考虑权重。

3.2.7 结论

从上述分析可以得出结论，尽管各模型在预测性能和训练时间上都有各自的优势，但在综合考虑预测精度和训练时间的情况下，ARIMA 模型成为本研究的首选。

首先，ARIMA 模型在所有的评价指标上，包括 MSE、RMSE、MAE、MAPE 和 SMAPE，均表现出最优的预测性能，这说明 ARIMA 模型在预测准确性上胜过其他模型。预测精度是我们选用预测模型的重要依据，因为高精度的预测结果可以为伸缩决策提供更可靠的依据，减小决策的风险。

其次，ARIMA 模型的训练时间仅为 0.1s，虽然不及 MA 模型的瞬时训练，但仍远低于 Prophet 模型的 4.7s，也比 VAR 模型的 0.4s 要短。在许多情况下，应当模型能够快速地进行训练和预测，以适应数据的实时变化和快速响应的需求。

因此，综合预测精度和训练时间，ARIMA 模型被我们选择为主要的预测模型。

第四章 算法的部署与应用

OpenFaaS 是一个开源的函数即服务（Function as a Service, FaaS）平台，它通过将业务逻辑封装成可重用、独立的函数来提供服务。相比于传统的服务器架构，FaaS 更加轻量级、弹性化，具有更好的可扩展性和更低的成本。

本章将介绍如何在 OpenFaaS 平台上部署本文设计的算法，并通过一个实际的案例来展示该算法的应用效果。

4.1 整体设计

本研究所提出的系统设计以高效性、稳定性和自适应性为目标，围绕算法模块、监控模块和控制模块进行布局。下面，我们将详细阐述每个模块的设计思路和实现方式。图 4-1 展示了整体设计思路。

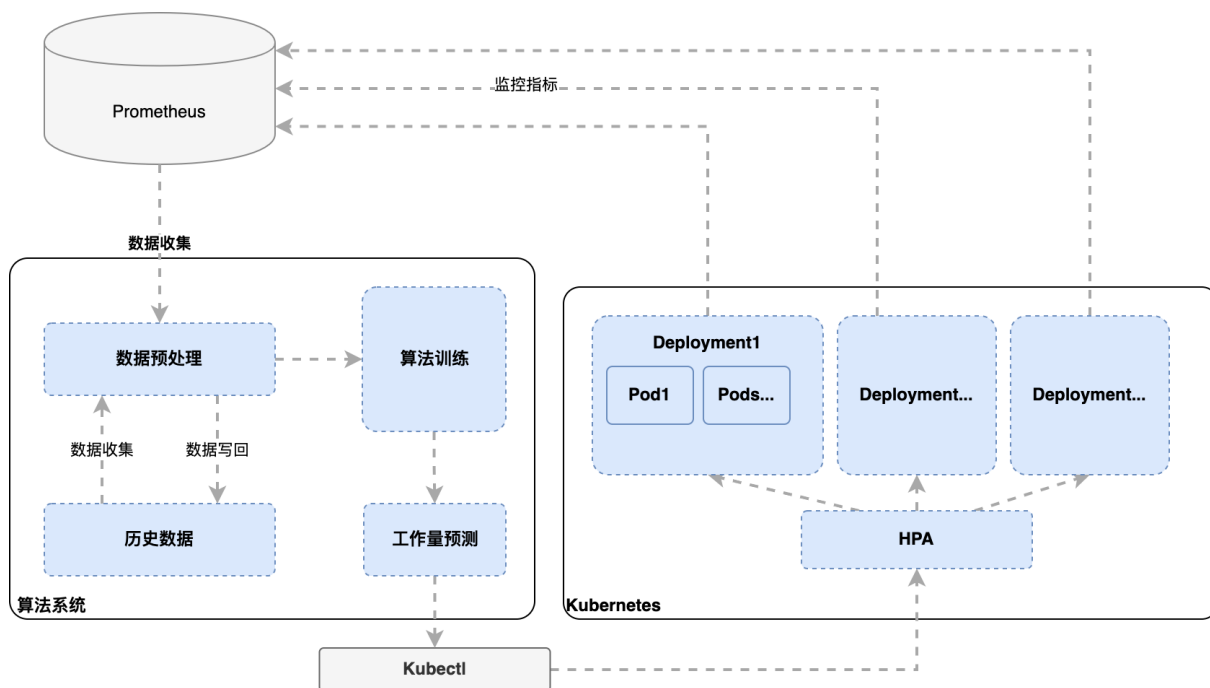


图 4-1 部署设计图

4.1.1 算法

算法模块是整个系统的核心，包括数据收集和训练两个子模块，负责模型的构建、训练和预测。该模块可通过手动调用或定时任务启动，对系统内的大量信息进行实时处理和决策，以实现自动化的负载调度和资源分配。

在系统启动之后，算法模块会通过调用 Prometheus 监控服务的 Web API 收集过去一段时间 T 的调用情况，并将这些数据与历史数据合并，构成训练样本，输入到训练子模块。经过一系列计算，模型在数秒内完成实时训练，并生成下一时间 $T+1$ 的预测值。

为了进一步实施负载调度,我们引入预定义的系数 k ,将预测值转换为下一时间 $T + 1$ 的副本数量。

4.1.2 监控

监控模块主要负责收集和存储系统的运行数据,提供给算法模块进行处理。这个模块由 Prometheus 提供,其指标来源于 OpenFaaS 容器。通过 Prometheus 的监控,我们可以对系统进行全方位的观察和分析,以便在必要时进行调整和优化。

4.1.3 控制

控制模块是系统的执行部分,负责将算法模块的决策转化为具体的行动。一旦算法模块完成计算并产生预测值,控制模块就会调用 `kubectl` 的有关命令,将预测值作为目标 Serverless 函数的最小副本数,进行伸缩。这样就可以根据实际需求,动态调整系统的负载和资源分配。

4.2 硬件与软件环境

该实验使用的操作系统为 Ubuntu 20.04.2 LTS x86_64,主机使用的是 OpenStack Nova 13.2.1-20230408163250_be95288。内核版本为 Linux 5.4.0-66-generic。CPU 为 Intel Xeon Gold 6278C (8 核),主频为 2.600GHz。物理内存大小为 32116MiB。GPU 使用的是 NVIDIA Tesla T4。

4.3 Kubernetes 集群的搭建和 OpenFaaS 的部署

为应用本算法,应当搭建一个 Kubernetes 集群,并在该集群中安装 OpenFaaS 以及本文设计的程序。

4.3.1 OpenFaaS 的部署

本节将介绍如何在本地部署 OpenFaaS 平台并测试其功能^[12]。

首先,应当安装 Docker 和 Kubernetes。

步骤 1: 首先,应当在本地环境中安装 Docker 和 Kubernetes 这两个相关的容器组件技术。

步骤 1.1: 要安装 Docker,应当按照 Docker 官方文档的指引,在 Linux 主机上执行以下命令,以便从 Docker 的软件源安装最新版本的 Docker Engine - Community:

```
1 curl -sLS https://get.docker.com | sudo sh
```

步骤 1.2: 然后,应当下载 Kubernetes 命令行工具 `kubectl` 的最新版本。可以按照 Kubernetes 官方文档的说明,在 Linux 主机上执行以下命令:

```

1 curl -LO "https://dl.k8s.io/release/$(curl -L -s \
    https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubect1"
2 sudo install -o root -g root -m 0755 kubect1 /usr/local/bin/kubect1
3 kubect1 version --client

```

步骤 2: 安装完 Docker 和 kubect1 之后, 应当安装 Kubernetes 的本地虚拟集群工具 Kind。可以从项目的 GitHub 发行版页面下载 Kind 的最新版本, 然后执行以下命令进行安装:

```

1 wget "https://github.com/kubernetes-sigs/kind/releases"\
2 "/download/v0.18.0/kind-linux-amd64" -O kind
3 chmod +x kind
4 sudo mv kind /usr/local/bin/

```

步骤 3: 使用 Kind 工具创建一个名为“openfaas”的 Kubernetes 本地集群:

```

1 kind create cluster --name openfaas

```

步骤 4: 我们继续执行 Kind 和 kubect1 命令设置当前工作环境变量以操作该 openfaas 集群:

```

1 export KUBECONFIG="$(kind get kubeconfig-path --name="openfaas")"

```

步骤 5: 最后, 我们使用 arkade 这一自动化工具安装 OpenFaaS 组件到集群之上。我们按照 arkade 的安装指引, 在 Linux 主机上下载并执行其安装脚本:

```

1 curl -sLS https://get.arkade.dev | sudo sh
2 arkade install openfaas

```

以上命令将自动安装所需的软件包并创建名为 openfaas 的 Kubernetes 集群, 最后使用 arkade 工具在集群中安装 OpenFaaS。安装完成后, 使用以下命令来检查 OpenFaaS 是否成功安装:

```

1 kubect1 get pods -n openfaas

```

可以看到, OpenFaaS 的各个组件均已正常运行。

4.3.1.1 部署函数

在 OpenFaaS 中, 函数是以 Docker 容器的形式进行部署和管理的。使用 faas-cli 工具可以方便地进行函数的部署。对于本文的示例, 可以使用以下命令将 nodeinfo 函数部署到 OpenFaaS 中:

```

1 faas-cli store deploy nodeinfo

```

这个命令的作用是从 OpenFaaS 的函数商店中拉取 nodeinfo 函数的镜像, 并将其部署到 OpenFaaS 环境中。

4.3.1.2 调用函数

函数成功部署到 OpenFaaS 环境后, 可以使用 HTTP 请求来调用。以下命令向部署在本地的 OpenFaaS 环境中的 nodeinfo 函数发送 HTTP GET 请求:

```
1 curl -v http://127.0.0.1:8080/function/nodeinfo
```

在接收到请求后, OpenFaaS 将会将请求转发给部署的 nodeinfo 函数, 并将函数的输出返回给客户端。

4.3.1.3 查看部署状态

使用 kubectl 命令可以查看 OpenFaaS 环境中部署的函数的状态。使用以下命令查看 nodeinfo 函数的部署状态^[13]:

```
1 kubectl get deployments -n openfaas-fn
```

命令返回一个包含了所有 OpenFaaS 环境中部署的函数的列表。对于本文的示例, 可以看到 nodeinfo 函数已经被成功部署。

此外, 也可以使用以下命令查看 nodeinfo 函数所在的 Pod 的状态:

```
1 kubectl get pods -n openfaas-fn
```

该命令将返回一个包含了所有 OpenFaaS 环境中正在运行的 Pod 的列表。对于本文的示例, 可以看到 nodeinfo 函数所在的 Pod 正在运行。

4.4 Prometheus 获取指标

Prometheus 提供了 HTTP API 来执行各种操作, 包括获取时间序列数据、元数据等^[14]。为了给算法提供数据, 应当使用 Prometheus 的 HTTP API 来获取 OpenFaaS 环境中的函数的相关指标。具体思路为, 获取最近一个小时的调用次数, 作为数据点加入到数据集中, 并快速训练出对应的 ARIMA 模型, 生成预测数据点, 然后根据预测数据点进行伸缩。

获取最近一小时的数据, 需使用 range queries API。

```
1 curl "http://localhost:9090/api/v1/query_range"\
2     "?query=gateway_function_invocation_total"\
3     "&start=<start_timestamp>"\
4     "&end=<end_timestamp>"\
5     "&step=<step_seconds>"
```

从而得到一个包含所需数据的响应:

```
1 {
2     "status": "success",
```



```

3   "data": {
4       "resultType": "matrix",
5       "result": [
6           {
7               "metric": {
8                   "__name__": "gateway_function_invocation_total",
9                   "instance": "localhost:9090",
10                  "job": "prometheus"
11              },
12              "values": [
13                  ...
14                  [1681347300, "220"]
15              ]
16          }
17      ]
18  }
19 }

```

4.4.1 根据预测结果调整副本数量

在 HTTP API 中, 可以使用 POST /system/functions 接口来部署或更新函数, 包括调整副本数量。为了调整 nodeinfo 函数的副本数量, 控制程序会发送以下 HTTP 请求

```

1   POST /system/functions HTTP/1.1
2   Host: <OpenFaaS Gateway URL>
3   Content-Type: application/json
4   Authorization: Bearer <Your OpenFaaS Token>
5
6   {
7       "service": "nodeinfo",
8       "image": "functions/nodeinfo",
9       "labels": {
10          "com.openfaas.scale.min": "2",
11          "com.openfaas.scale.max": "5"
12      }
13  }

```

在 labels 字段中, com.openfaas.scale.min 和 com.openfaas.scale.max 分别表示最小和最大的副本数量。

除此之外, 也可以通过 kubectl 直接设置副本的数量:

```

1   kubectl scale deployment nodeinfo -n openfaas-fn --replicas=5

```

然后, OpenFaaS 控制器将会根据副本数量的变化, 自动调整 `nodeinfo` 函数的副本数量。可以通过如下命令检查 `nodeinfo` 函数的副本数量:

```
1 kubectl get deployment nodeinfo -n openfaas-fn -o=jsonpath='{.spec.replicas}'
```

4.4.2 自动伸缩: HPA 控制器

在容器编排中, HPA (Horizontal Pod Autoscaler) 用于根据负载自动调整 Pod 的数量。Deployment 是用于定义和管理 Pod 副本的资源对象。Pod 则代表着运行在 Kubernetes 集群中的一个实例化容器^[15]。

图 4-2 中的箭头表示了对象之间的关系。箭头从 HPA 指向 Deployment, 表示 HPA 通过扩展或收缩 Deployment 来调整 Pod 的数量。Deployment 和 Pod 之间的箭头表示 Deployment 控制着多个 Pod 的创建和管理。

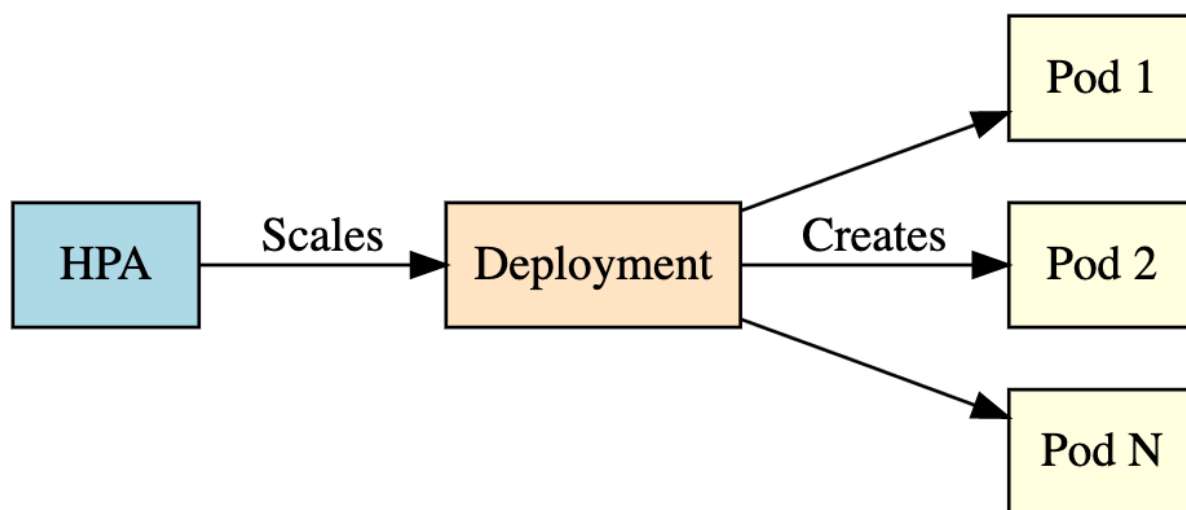


图 4-2 HPA 控制器原理

4.5 流量模拟与伸缩实验

4.5.1 请求变化曲线的构造

为了验证自动伸缩, 首先需要生成流量的变化曲线。由于实际使用时伸缩的周期是数小时, 为了方便验证, 我们将时间尺度缩小到数分钟。然后, 我们选用一个简单的函数来模拟请求流量的变化曲线。

具体而言, 这个函数充当一个数据生成器, 它可以根据给定的种子和长度, 生成具有特定属性的时间序列数据。我们希望这些数据具有一定的随机性, 以模拟实际场景的不确定性; 同时, 我们还希望它具有一定的周期性 (包括短周期和长周期), 以反映某些周期性因素的影响; 此外, 我们希望这些数据具有整体的上升趋势, 从而反映某些长

期趋势（例如业务增长）的影响。最后，我们希望这些数据是整数，并且其值在 0 1000 之间，从而确保它们与实际的函数调用次数具有相同的数量级。

数据生成器函数的代码如下：

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def generate_data(seed, t_length, small_period=24, large_period=720):
5     np.random.seed(seed)
6
7     # 初始化序列
8     y = np.zeros(t_length)
9
10    # 生成随机步长。此处，我们使步长的平均值稍微偏正，以保证整体的升趋势。
11    steps = np.random.randint(-40, 45, size=t_length - 1)
12
13    # 计算随机漫步序列
14    for t in range(1, t_length):
15        y[t] = y[t-1] + steps[t-1]
16
17    # 添加周期性趋势
18    t = np.arange(t_length)
19    y += np.sin(2 * np.pi * t / small_period) * 50 # 小周期趋势
20    y += np.sin(2 * np.pi * t / large_period) * 100 #
        大周期趋势，可以调整振幅以增加或减少周期性趋势的强度
21
22    # 为了保证y的范围在0~1000之间，我们将y线性映射到这个范围
23    y = (y - np.min(y)) / (np.max(y) - np.min(y)) * 1000
24
25    # 因为y应该是整数，所以我们进行四舍五入
26    y = np.round(y).astype(int)
27
28    return y

```

它的工作原理如下：

首先，我们使用给定的种子初始化随机数生成器，以确保本研究的数据生成过程是可复现的。然后，我们生成一个随机步长序列，这个步长序列可以是正的或负的，但我们使其平均值稍微偏正，以保证整体的上升趋势。然后，我们使用这个步长序列生成一个随机漫步序列，这个随机漫步序列体现了数据的随机性。

接下来，我们添加了两个周期性趋势。一个是小周期趋势，周期为 24，模拟了一天内的变化；另一个是大周期趋势，周期为 720，模拟了一个月内的变化。我们使用 sine 函数来生成这两个周期性趋势，因为 sine 函数是周期性的，而且它的值在 -1 和 1 之间，

这使得可以通过调整振幅来控制周期性趋势的强度。

最后,为了保证生成的数据的值在 0 1000 之间且是整数,我们将随机漫步序列和周期性趋势的和线性映射到这个范围,然后进行四舍五入。线性映射的过程是通过计算最小值和最大值,然后使用这两个值将数据缩放和平移到目标范围。

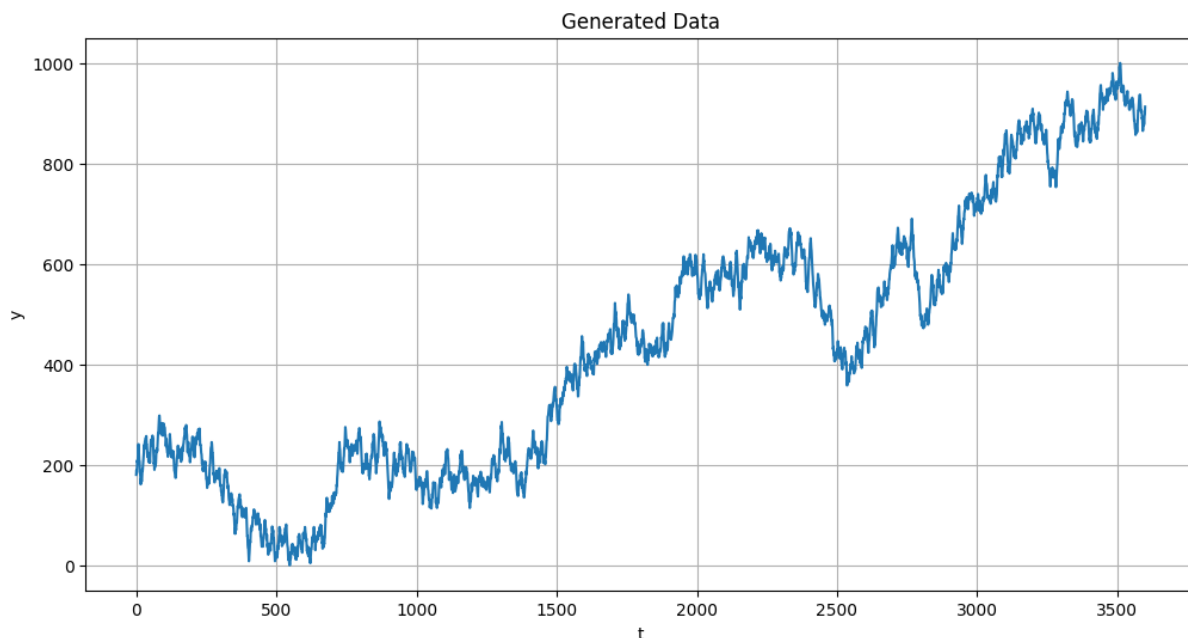


图 4-3 生成的数据示例

生成的数据的示例如图 4-3 所示。

4.5.2 基于模拟流量的自动伸缩仿真

我们使用 0 到 999 时刻的数据作为初始历史数据,输入到 ARIMA 模型训练,并预测 1000 时刻的访问量。从 1000 时刻开始,每隔 30 秒,采用下一个时刻的数据点作为模拟的请求数量。并滑动历史数据窗口,将新的数据点加入到模型训练的历史数据集中,以模拟实际的工作负载预测过程。

基于预测访问量,使用请求/容器数量的比例(记作 k)来计算需要扩容至的容器数量。此处我们采用 $k=100$,也即每个容器每秒处理 100 个请求的假设。此假设仅用于演示,其取值不影响预测性能或伸缩效率。在实际应用时,应当根据服务器配置情况测量得到此参数。

图 4-4 展示访问量与副本数量的关系,即假设伸缩不消耗时间情况下副本的数量关系。

4.5.3 仿真结果与结果分析

使用上述结果进行仿真,对于预测区间 [1001,2000], ARIMA 模型得到的结果如图 4-5 所示。作为基准的 MA (WIN_SIZE=3) 模型得到的结果如图 4-6 所示。这两张图

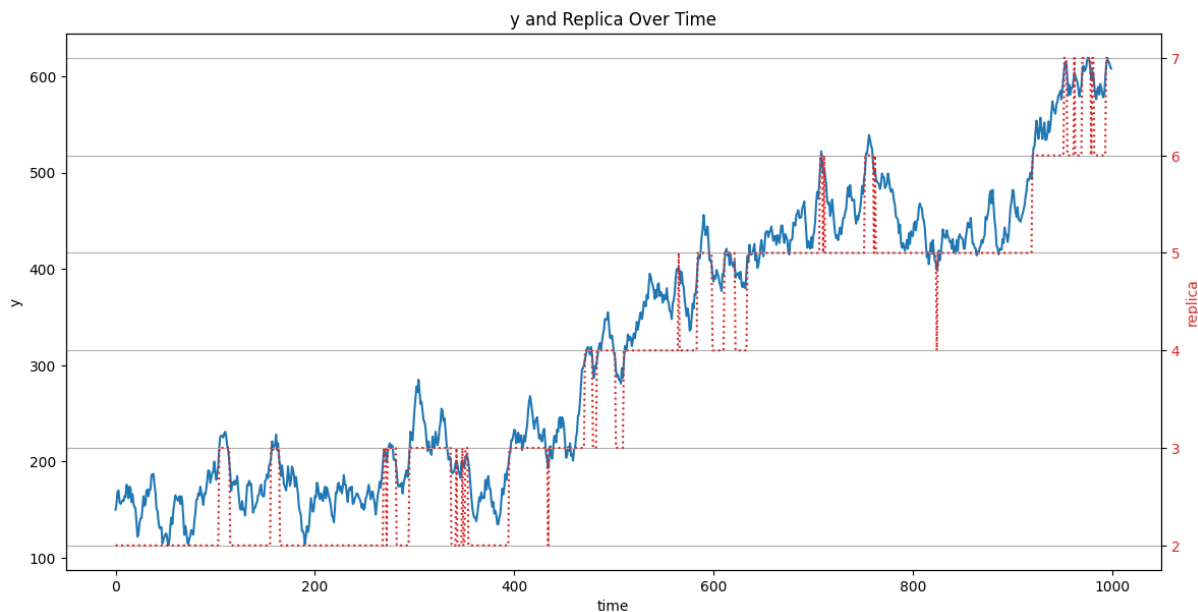


图 4-4 访问量与副本数量

展示实际值（蓝色曲线）与预测值（绿色曲线）之间的关系，同时使用右侧坐标轴展示实际所需副本数和预测所需副本数之间的关系。

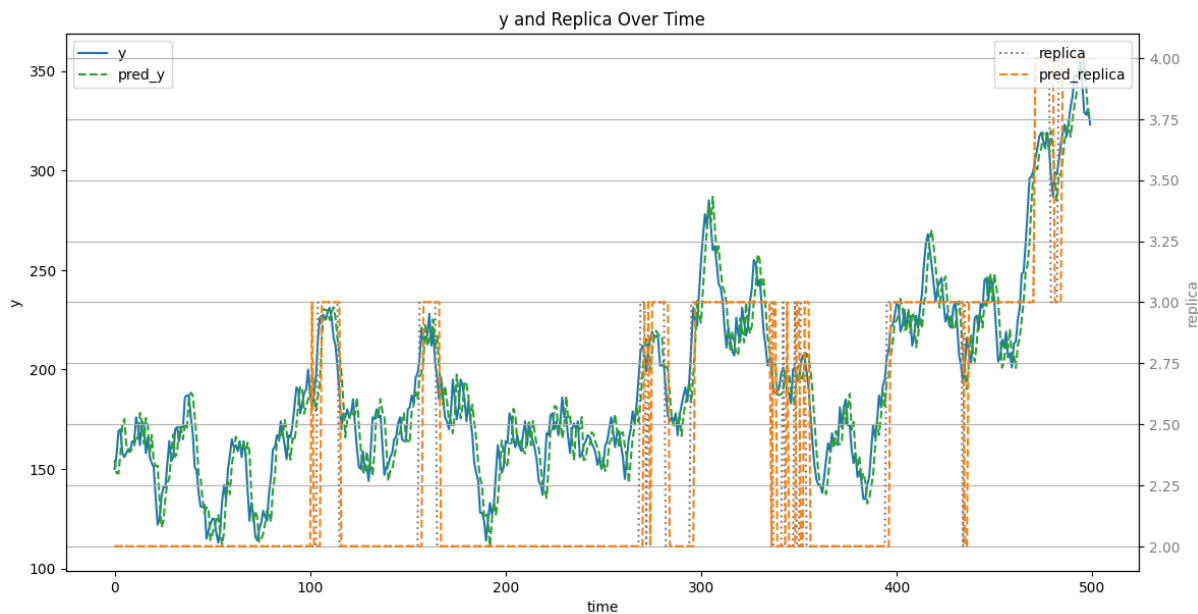


图 4-5 ARIMA 模型仿真结果

表 4-1 展示模型的性能指标。从 MAPE、SMAPE 的结果来看，模型的预测性能较好。且 ARIMA 模型的性能在所有指标上超过 MA 模型。

而副本数量预测任务，由于相当于对访问量预测进行向上取整，预测的性能表现更为优异，如表 4-2 所示。副本数量的预测几乎完全准确。且 ARIMA 模型的性能在所有指标上均优于 MA 模型。

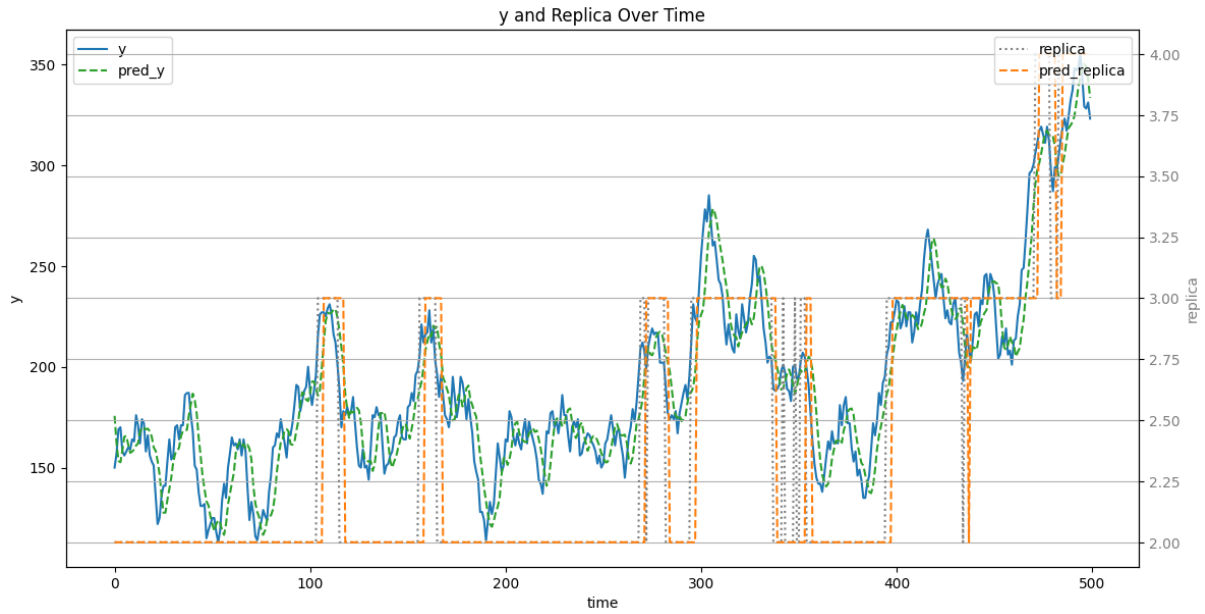


图 4-6 MA 模型仿真结果

表 4-1 基于仿真数据预测的模型性能

指标	数值	MA 模型基准
MSE	169.09	243.23
RMSE	13.00	15.60
MAE	10.68	12.72
MAPE	5.78 %	6.89%
SMAPE	5.78 %	6.87%

表 4-2 基于仿真数据预测的模型性能 (副本数量)

指标	数值	MA 模型基准
MSE	0.074	0.086
RMSE	0.27	0.29
MAE	0.074	0.086
MAPE	2.93 %	3.30%
SMAPE	2.86 %	3.28%

第五章 结论与展望

5.1 结论

5.1.1 本文的主要成果

本研究中提出并实施了一种基于负载预测的 Serverless 计算的自动伸缩算法。主要发现如下：

- 我们基于 MA、VAR、Prophet 和 ARIMA 模型实现了有效的流量预测机制，从而能够在工作负载发生变化之前做出响应，提前进行自动伸缩。通过对实际的 Serverless 工作负载进行实验，我们发现这些模型能够有效预测其变化趋势。
- 我们设计并实施了一套基于 Prometheus 的自动预警机制。通过收集 Serverless 应用的监控指标，并将这些数据输入到预测模型中，可以提前做出伸缩决策，从而避免因工作负载变化导致的性能下降或资源浪费。
- 我们还实现了函数副本数量的自动伸缩。通过调整副本数量，Serverless 应用可以根据实际的工作负载需求动态调整资源使用，从而提高资源利用率和运行稳定性。

值得一提的是，我们在每个预测时刻之前，会采用滑动窗口的方式重新训练模型，并且自动选择最优的 p, d, q 参数。这主要有两点好处：一是可以避免历史数据膨胀导致预测的时间复杂度常数过高；二是可以避免模型参数无法同步于历史数据的变化导致预测结果的不准确性。

5.1.2 对实际问题的解决方案

本研究所提出的自动伸缩算法和方案，为如何实现 Serverless 计算的自动伸缩提供了有效的解决方案。通过自动预警、负载预测、自动伸缩的结合，可以在工作负载变化之前做出响应，动态调整函数副本数量，提高资源利用率和运行稳定性。

5.2 局限性与未来工作

5.2.1 本文的不足之处

虽然本研究取得了一些重要的发现，但还存在以下不足之处：

- 本研究的负载预测模型基于历史数据进行预测，但对于那些历史数据不足，预测准确性可能会降低。
- 我们发现多数的 Serverless 应用，即便在日尺度下有非常高的访问次数，但在分钟或秒的尺度之下，统计周期内可能根本没有访问，所以对于实际场景，精确到秒的预测的难度会非常大。这也是我们选择以小时为单位的原因。

- 允许零副本的情况下，本研究的自动伸缩算法由于采用向上取整，最少的副本数量为 1，这可能会导致资源浪费。
- 我们实验时发现特定时间点负载的变化幅度较大，导致频繁的伸缩行为，这可能会影响应用的性能。

5.2.2 可进一步探讨和优化的方向

针对上述不足之处，未来的研究可以从以下方向进行：

- **使用预训练模型进行预测。**可以考虑使用 Transformer 等模型，在同类 Serverless 应用上进行预训练，再使用具体的应用进行微调，以提高预测的准确性^[16]。
- **增加更多的预测特征。**目前主要针对时间和访问次数进行预测，未来可以考虑引入更多内部和外部的特征，例如，CPU 使用率，内存使用率，网络延迟等，以提高预测的准确性。不过这主要适用于多特征的预测模型。
- **考虑支持零副本的缩容。**对于重要度不高，能接受数秒的启动延迟的应用，可以考虑支持零副本的缩容，从而避免资源浪费。
- **考虑增加对临界值的特殊处理。**预测值量变临界情况下，可以增加一定的迟滞性，从而避免过于频繁的伸缩。

这些方向将为我们提供更深入的理解和优化 Serverless 计算的自动伸缩。

参考文献

- [1] Jumpertz Oliver. How AWS Lambda Works Under The Hood [EB/OL]. 2020. <https://oliverjumpertz.com/how-aws-lambda-works-under-the-hood/>.
- [2] OpenFaaS Org. OpenFaaS Design & Architecture: Gateway [EB/OL]. 2023 [2023-4-10]. <https://docs.openfaas.com/architecture/gateway/>.
- [3] Rizkya Indah, Syahputri Khalida, Sari Rachida et al. Autoregressive Integrated Moving Average (ARIMA) Model of Forecast Demand in Distribution Centre [J]. IOP Conference Series Materials Science and Engineering. 598 (1). 2019, September: 012–071.
- [4] Hyndman Rob J, Athanasopoulos George. Forecasting: Principles and Practice [M]. OTexts, 2018.
- [5] Taylor Sean J, Letham Benjamin. Forecasting at Scale [J]. The American Statistician. 72 (1). 2018: 37–45.
- [6] Robert Christian, Casella George. A Short History of Markov Chain Monte Carlo: Subjective Recollections from Incomplete Data [J/OL]. Statistical Science. 26 (1). 2011: 102–115. <http://www.jstor.org/stable/23059158>.
- [7] Tran Minh-Ngoc, Nguyen Trong-Nghia, Dao Viet-Hung. A practical tutorial on Variational Bayes. 2021.
- [8] Akaike H. This Week’s Citation Classic [J]. Current Contents Engineering, Technology, and Applied Sciences. 12 (51). 1981: 42. Hirotogu Akaike comments on how he arrived at AIC.
- [9] Schwarz Gideon E. Estimating the dimension of a model [J]. Annals of Statistics. 6 (2). 1978: 461–464. MR 0468014.
- [10] Niedźwiecki Maciej, Ciołek Marcin. Akaike’s final prediction error criterion revisited [C]. In 2017 40th International Conference on Telecommunications and Signal Processing (TSP). 2017 : 237–242.
- [11] Pollock D S G. Handbook of Time Series Analysis, Signal Processing, and Dynamics [M]. har/cdr ed. Academic Press, 1999.
- [12] Ellis Alex. Get started with OpenFaaS and KinD [EB/OL]. 2019. <https://blog.alexellis.io/get-started-with-openfaas-and-kind/>.
- [13] Kubernetes Authors. Kubernetes Deployment [EB/OL]. 2023. <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>.
- [14] Prometheus Authors. Prometheus Querying API [EB/OL]. 2023. <https://prometheus.io/docs/prometheus/latest/querying/api/>.
- [15] Kubernetes Authors. Kubernetes HPA [EB/OL]. 2023. <https://kubernetes.io/zh-cn/docs/tasks/run-application/horizontal-pod-autoscale/>.
- [16] Wu Neo, Green Bradley, Ben Xue et al. Deep Transformer Models for Time Series Forecasting: The Influenza Prevalence Case [J]. arXiv preprint arXiv:2001.08317. 2020: 1.

致 谢

首先,我要衷心地感谢我的女朋友。在我写作此论文的过程中,她给予了我巨大的支持和帮助。她不仅陪伴我熬夜讨论文章结构,她还细心地帮我检查文章中的错误和不足之处,提出宝贵的修改意见,这使得这篇论文的完成离不开她的帮助。亲爱的,谢谢你在这段日子里给予我的理解,鼓励和支持,让我有动力完成这篇论文。

其次,我要感谢我的父母。在我的生命中,父母一直给予我无条件的爱和支持,鼓励我去追求自己的梦想。在我学习实践并完成此论文的过程中,父母也一直在我身后支持和理解我。阅读了此论文初稿后,父母也提出宝贵的意见和建议,帮助我进一步改进文章。亲爱的父母,我能有今天的成就离不开你们年复一年的培养,谢谢你们!

再次,我要特别感谢我的导师和老师。导师在我选择论文题目、开展研究以及完成论文的每个阶段都给予我悉心的指导,不辞辛苦地检查我的论文并提出修订意见,这些都使我受益匪浅。我也特别感谢其他老师在我学习期间对我的帮助和培养,使我有机会完成此论文。

最后但同样重要的是,我要感谢我的同学和朋友。你们的陪伴和鼓励让我在完成论文的日子里倍感温暖。你们对我提出的问题和困惑也都给予详细的解答和意见,这让我在论文的思路和研究方法上有很大的收获。亲爱的同学和朋友,我会永远记住你们在这段日子里给予我的支持和帮助。

再一次,感谢所有在我完成这篇论文过程中给予帮助和支持的人。你们的鼓励和理解是我完成此篇论文的动力之源。谢谢你们!祝你们身体健康、工作顺利!

Deep Transformer Models for Time Series Forecasting: The Influenza Prevalence Case

Neo Wu¹ Bradley Green¹ Xue Ben¹ Shawn O'Banion¹

Abstract

In this paper, we present a new approach to time series forecasting. Time series data are prevalent in many scientific and engineering disciplines. Time series forecasting is a crucial task in modeling time series data, and is an important area of machine learning. In this work we developed a novel method that employs Transformer-based machine learning models to forecast time series data. This approach works by leveraging self-attention mechanisms to learn complex patterns and dynamics from time series data. Moreover, it is a generic framework and can be applied to univariate and multivariate time series data, as well as time series embeddings. Using influenza-like illness (ILI) forecasting as a case study, we show that the forecasting results produced by our approach are favorably comparable to the state-of-the-art.

1. Introduction

Seasonal influenza epidemics create huge health and economic burdens, causing 291,000 - 646,000 deaths worldwide (Iuliano et al., 2018). In the United States, the Centers for Disease Control and Prevention (CDC) publishes weekly ILI reports based on its surveillance network. Despite its significance in monitoring disease prevalence, typically there is at least a one-week delay of ILI reports due to data collection and aggregation. Therefore, forecasting ILI activity is critical for real-time disease monitoring, and for public health agencies to allocate resources to plan and prepare for potential pandemics.

A variety of methods have been developed to forecast these ILI time series data. These approaches range from mechanistic approaches to statistical and machine learning meth-

ods. Mechanistic modeling is based on the understanding of underlying disease infection dynamics. For example, compartmental methods such as SIR are popular approaches to simulating disease spreading dynamics.

Statistical and machine learning methods leverage the ground truth data to learn the trends and patterns. One popular family of methods includes auto-regression (AR), autoregressive moving average (ARMA), and autoregressive integrated moving average (ARIMA). Additionally, deep learning approaches based on convolutional and recurrent neural networks have been developed to model ILI data. These sequence-aligned models are natural choices for modeling time series data. However, due to “gradient vanishing and exploding” problems in RNNs and the limits of convolutional filters, these methods have limitations in modeling long-term and complex relations in the sequence data.

In this work, we developed a novel time series forecasting approach based on Transformer architecture (Vaswani et al., 2017). Unlike sequence-aligned models, Transformer does not process data in an ordered sequence manner. Instead, it processes entire sequence of data and uses self-attention mechanisms to learn dependencies in the sequence. Therefore, Transformer-based models have the potential to model complex dynamics of time series data that are challenging for sequence models. In this work we use ILI forecasting as a case study to show that a Transformer-based model can be successfully applied to the task of times series forecasting and that it outperforms many existing forecasting techniques. Specifically, our contributions are the following:

- We developed a general Transformer-based model for time series forecasting.
- We showed that our approach is complementary to state space models. It can model observed data. Using embeddings as a proxy, our approach can also model state variables and phase space of the systems.
- Using ILI forecasting as a case study, we demonstrated that our Transformer-based model is able to accurately forecast ILI prevalence using a variety of features.
- We showed that in the ILI case our Transformer-based model achieves state-of-the-art forecasting results.

¹Google, LLC, 651 N 34th St., Seattle, WA 98103 USA. Correspondence to: Neo Wu <neowu@google.com>, Bradley Green <brg@google.com>, Xue Ben <sherryben@google.com>, Shawn O'Banion <obanion@google.com>.

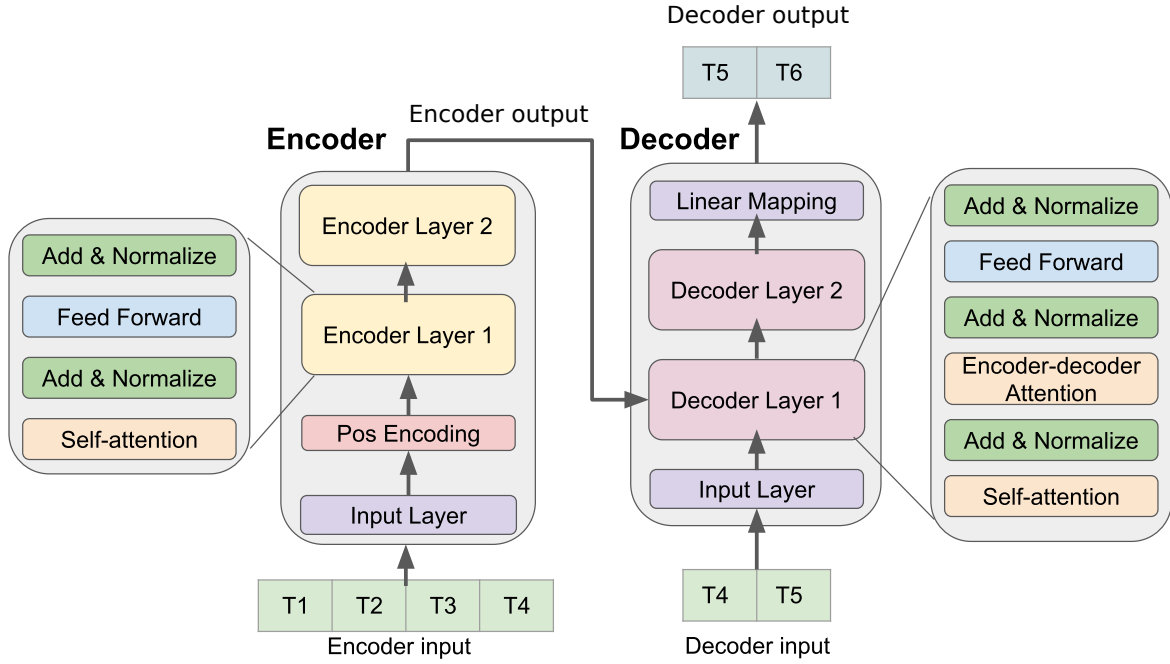


Figure 1. Architecture of Transformer-based forecasting model.

2. Related Work

Several studies have used Internet data such as Google Trends (Ginsberg et al., 2009), Twitter (Paul et al., 2014) and Wikipedia (McIver & Brownstein, 2014) to forecast ILI ratios. Google Flu Trends (GFT) employs a linear model that uses Google search volumes of predefined terms to estimate current ILI ratios (“nowcasting”). While initially regarded as a great success, GFT suffered from over-estimation of peak ILI magnitude in subsequent years (Olson et al., 2013; Lazer et al., 2014).

Other studies combined GFT with new modeling techniques and additional signals. Lazer et al (2014) suggested an autoregression-based (AR) approach to extend GFT. Santilana et al (2014) improved GFT by developing a model that automatically selects queries and updates models for ILI forecasting. Araz et al (2014) built linear regression models using GFT data with extra signals. Yang et al (2015) developed an autoregression-based model with Google search data (“ARGO”) to estimate influenza epidemics. ARGO outperforms previous Google-search-based models. More recently, a new ensemble model (“ARGONet”) built on top of ARGO was developed (Lu et al., 2019). This new approach leverages spatial information to improve the model and achieved state-of-the-art results for ILI forecasting.

Deep learning techniques have also been used for ILI fore-

casting. Liu et al (2018) trained an LSTM-based model to predict influenza prevalence using Google Trends, climate, air pollution and virological surveillance data. Venna et al (2019) developed an LSTM-based multi-stage model to incorporate climate and spatio-temporal adjustment factors for influenza forecasting. Attention-based techniques are also applied for ILI forecasting. Zhu et al (2019) developed multi-channel LSTM neural networks to learn from different types of inputs. Their model uses an attention layer to associate model output with the input sequence to further improve forecast accuracy. Kondo et al (2019) adapted a sequence-to-sequence (“Seq2Seq”) model with a similar attention mechanism to predict influenza prevalence and showed that their approach outperformed ARIMA and LSTM-based models.

3. Background

3.1. Influenza and ILI

Influenza is a common infectious disease caused by infection of influenza virus. Influenza affects up to 35 million people each year and creates huge health and economic burdens (Iuliano et al., 2018; Putri et al., 2018). In the United States, the Centers for Disease Control and Prevention (CDC) coordinates a reporting network over a large geographic area. Participating health providers within the

network report statistics of patients showing influenza-like illness (ILI) symptoms. ILI symptoms are usually defined as *fever and cough and/or sore throat*. The ILI ratio is computed as the ratio of the number of patients seen with ILI and the total number of patient visits that calendar week. The CDC published ILI ratios for the USA and all but one individual state (Florida). Additionally, state-level ILI ratios are normalized by state populations.

3.2. State Space Models

State space modeling (SSM) is widely applied to dynamical systems. The evolution of dynamical systems is controlled by non-observable state variables. The system exhibits observable variables which are determined by state variables. SSM has been applied to study complex systems in biology and finance.

State space models model both state and observable variables. For example, a generalized linear state space model can be expressed in the following equations:

$$\begin{aligned} x_t &= Z_t \alpha_t + \epsilon_t \\ \alpha_{t+1} &= T_t \alpha_t + R_t \eta_t, t = 1, \dots, n, \end{aligned} \quad (1) \quad (2)$$

where x_t and α_t are time-indexed observation vectors and state vectors, respectively. Equation 1, called observation equation, is a regression-like equation. It models the relationship of observable x_t and the underlying state variable α_t . Equation 2 is the state equation, and has autoregressive nature. It governs how the state variables evolve over time. ϵ_t and η_t are innovation components and are usually modeled as Gaussian processes.

In this section, we briefly mention a few commonly used SSM models in ILI forecasting.

Compartmental Models Compartmental models are a specific form of SSMs and have been widely used to study infectious diseases. In a compartmental model, a population is divided into different groups (“compartments”). Each group is modeled by a time-dependent state variable. A prominent example of compartmental model is “Suscepted-Infected-Recovered” (SIR) model, where the system is governed by three state variables ($S(t)$, $I(t)$, $R(t)$) through the following ordinary differential equations:

$$\begin{aligned} \frac{dS}{dt} &= -\frac{\beta IS}{N} \\ \frac{dI}{dt} &= \frac{\beta IS}{N} - \gamma I \\ \frac{dR}{dt} &= \gamma I \end{aligned}$$

In this treatment, ILI time series is an observable variable of the system: $ILI(t) = I(t)/(I(t) + S(t) + R(t))$.

Although originally developed to model infectious diseases, compartmental models have been applied to other disciplines such as ecology and economics. While compartmental models are useful, they require prior knowledge on the parameters of the differential equations and they lack flexibility of updating parameters upon new observations.

ARIMA Box-Jenkins ARIMA (Auto-Regressive Integrated Moving Average) is another popular approach to modeling dynamical systems. ARIMA models the observed variable x_t and assumes x_t can be decomposed into trend, seasonal and irregular components. Instead of modeling these components separately, Box and Jenkins had the idea of differencing the time series x_t in order to eliminate trend and seasonality. The resulting series is treated as stationary time series data and is modeled using combination of its lagged time series values (“AR”) and moving average of lagged forecast errors (“MA”). An ARIMA model is typically specified by a tuple (p, d, q) , where p and q define the orders of AR and MA, and d specifies the order of differencing operation.

ARIMA can be written in SSM form, and common SSM techniques such as filtering and smoothing can be applied to ARIMA as well. Nevertheless, ARIMA is a kind of “blackbox” approach where the model purely depends on the observed data and has no analysis of the states of the underlying systems (Durbin & Koopman, 2012).

Time Delay Embedding For a scalar time series data x_t , its time delay embedding (TDE) is formed by embedding each scalar value x_t into a d -dimensional time-delay space:

$$TDE_{d,\tau}(x_t) = (x_t, x_{t-\tau}, \dots, x_{t-(d-1)\tau})$$

For any non-linear dynamical systems, the delay-embedding theorem (Takens’ theorem) (Takens, 1981) states that there exists a certain (d, τ) -time delay embedding such that the evolution of the original state variables (“phase space”) can be recovered in the delay coordinates of the observed variables. In the case of ILI forecasting, Takens’ theorem suggests that $TDE_{d,\tau}$ of ILI ratios (“observed variable”) can approximate the underlying dynamical systems governed by biological and physical mechanisms.

TDEs were first explored for time series forecasting in the seminal work by Sugihara and May (1990). They showed that TDEs can be used to make short-range predictions based on the qualitative assessment of a system’s dynamics without any knowledge on the underlying mechanisms. They developed two TDE-based models to predict chickenpox and measles prevalence, and compared them with AR-based approaches. Their analysis suggests that TDE-based model performs equally well for chickenpox case prediction and outperforms AR for measles case prediction.

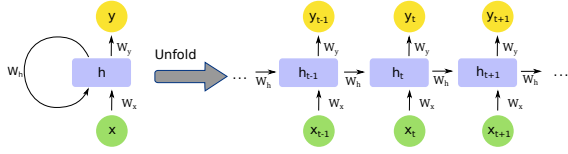


Figure 2. Folded and unfolded representations of recurrent neural network.

In the SSM framework, the time delay embedding is a powerful tool to bridge state variables and observed data by making it possible to learn geometrical and topological information of underlying dynamical systems without an understanding of the state variables and phase space of the systems. Despite the amazing property of TDEs, to the best of our knowledge, TDEs haven't been studied extensively for machine learning models.

3.3. Sequence Models

Many real-world machine learning tasks deal with different types of sequential data ranging from natural language text, audio, and video, to DNA sequences and time series data. Sequence models are specifically designed to model such data. In this section, we briefly review a few different types of common sequence models.

Recurrent Neural Networks Unlike traditional feed-forward networks, RNN is recurrent in nature - it performs the same function to each input x_t , and the output y_t depends on both the input x_t and the previous state h_{t-1} .

The simple RNN illustrated in Figure 3.3 can be expressed as follows:

$$\begin{aligned} h_t &= \sigma(W_x x_t + W_h h_{t-1} + b_h) \\ y_t &= \sigma(W_y h_t + b_y) \end{aligned}$$

Where x_t is the input vector, h_t is the hidden state vector, and y_t is the output vector. W 's and b 's are learned parameters, and σ is the activation function.

LSTM While RNN has internal memory to process sequence data, it suffers from gradient vanishing and exploding problems when processing long sequences. Long Short-Term Memory (LSTM) networks were specifically developed to address this limitation (Hochreiter & Schmidhuber, 1997). LSTM employs three gates, including an input gate, forget gate and output gate, to modulate the information flow across the cells and prevent gradient vanishing and explosion.

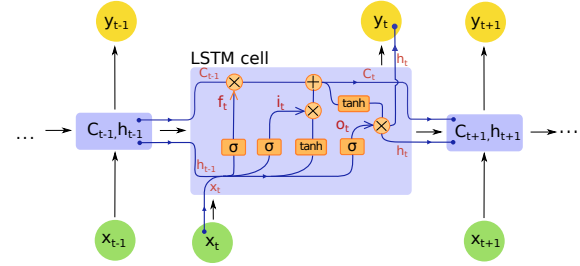


Figure 3. Long Short-Term Memory network and LSTM unit.

$$\begin{aligned} f_t &= \sigma(W_f[h_{t-1}, x_t] + b_f) \\ i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= \tanh(W_C[h_{t-1}, x_t] + b_C) \\ C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\ y_t &= \sigma(W_y[h_{t-1}, x_t] + b_y) \\ h_t &= y_t * \tanh(C_t) \end{aligned}$$

Seq2Seq Sequence-to-sequence (Seq2Seq) architecture is developed for machine learning tasks where both input and output are sequences. A Seq2Seq model is comprised of three components including an encoder, an intermediate vector, and a decoder. Encoder is a stack of LSTM or other recurrent units. Each unit accepts a single element from the input sequence. The final hidden state of the encoder is called the encoder vector or context vector, which encodes all of the information from the input data. The decoder is also made of a stack of recurrent units and takes the encoder vector as its first hidden state. Each recurrent unit computes its own hidden state and produces an output element. Figure 3.3 illustrates the Seq2Seq architecture.

Seq2Seq has been widely applied in language translation tasks. However, its performance degrades with long sentences because it cannot adequately encode a long sequence into the intermediate vector (even with LSTM cells). Therefore, long-term dependencies tend to be dropped in the encoder vector.

4. Model

4.1. Problem Description

We formulate ILI forecasting as a supervised machine learning task. Given a time series containing N weekly data points $x_{t-N+1}, \dots, x_{t-1}, x_t$, for M -step ahead prediction, the input X of the supervised ML model is $x_{t-N+1}, \dots, x_{t-M}$, and the output Y is $x_{t-M+1}, x_{t-M+2}, \dots, x_t$. Each data point x_t can be a scalar or a vector containing multiple features.

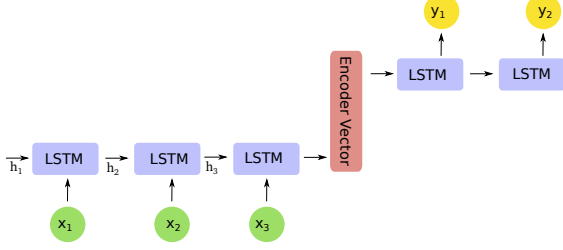


Figure 4. Sequence-to-sequence (Seq2Seq) architecture to model sequence input and output.

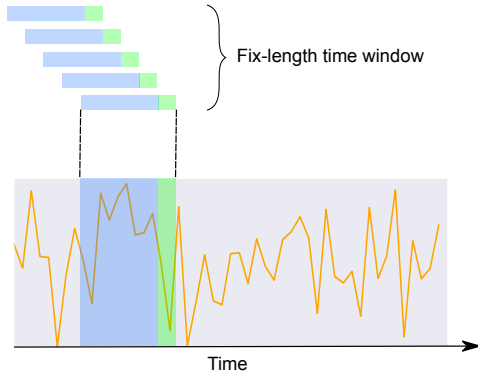


Figure 5. Using sliding window to construct supervised learning examples from time series data.

4.2. Data

We utilized country- and state-level historical ILI data from 2010 to 2018 from the CDC (CDC).

To produce a labeled dataset, we used a fixed-length sliding time window approach (Figure 5) to construct X, Y pairs for model training and evaluation. Before applying the sliding window to get features and labels, we perform min-max scaling on all the data with the maximum and minimum values of training dataset. We then run a sliding window on the scaled training set to get training samples with features and labels, which are the previous N and next M observations respectively. Test samples are also constructed in the same manner for model evaluation. The train and test split ratio is 2:1. Training data from different states are concatenated to form the training set for the global model.

4.3. Transformer Model

4.3.1. MODEL ARCHITECTURE

Our Transformer-based ILI forecasting model follows the original Transformer architecture (Vaswani et al., 2017) consisting of encoder and decoder layers.

Encoder The encoder is composed of an input layer, a positional encoding layer, and a stack of four identical encoder layers. The input layer maps the input time series data to a vector of dimension d_{model} through a fully-connected network. This step is essential for the model to employ a multi-head attention mechanism. Positional encoding with sine and cosine functions is used to encode sequential information in the time series data by element-wise addition of the input vector with a positional encoding vector. The resulting vector is fed into four encoder layers. Each encoder layer consists of two sub-layers: a self-attention sub-layer and a fully-connected feed-forward sub-layer. Each sub-layer is followed by a normalization layer. The encoder produces a d_{model} -dimensional vector to feed to the decoder.

Decoder We employ a decoder design that is similar to the original Transformer architecture (Vaswani et al., 2017). The decoder is also composed of the input layer, four identical decoder layers, and an output layer. The decoder input begins with the last data point of the encoder input. The input layer maps the decoder input to a d_{model} -dimensional vector. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer to apply self-attention mechanisms over the encoder output. Finally, there is an output layer that maps the output of last decoder layer to the target time sequence. We employ look-ahead masking and one-position offset between the decoder input and target output in the decoder to ensure that prediction of a time series data point will only depend on previous data points.

4.3.2. TRAINING

Training Data and Batching In a typical training setup, we train the model to predict 4 future weekly ILI ratios from 10 trailing weekly datapoints. That is, given the encoder input $(x_1, x_2, \dots, x_{10})$ and the decoder input (x_{10}, \dots, x_{13}) , the decoder aims to output (x_{11}, \dots, x_{14}) . A look-ahead mask is applied to ensure that attention will only be applied to datapoints prior to target data by the model. That is, when predicting target (x_{11}, x_{12}) , the mask ensures attention weights are only on (x_{10}, x_{11}) so the decoder doesn't leak information about x_{12} and x_{13} from the decoder input. A minibatch of size 64 is used for training.

Optimizer We used the Adam optimizer (Kingma & Ba, 2015) with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$. A custom learning rate with following schedule is used:

$$lrate = d_{model}^{0.5} * \min(step_num^{0.5}, step_num * warmup_steps^{-1.5})$$

Where $warmup_steps = 5000$.

Regularization We apply dropout techniques for each of the three types of sub-layers in the encoder and decoder: the

self-attention sub-layer, the feed-forward sub-layer, and the normalization sub-layer. A dropout rate of 0.2 is used for each sub-layer.

4.3.3. EVALUATION

In evaluation, labeled test data are constructed using a fixed-length sliding window as well. One-step ahead prediction is performed by the trained Transformer model. We computed Pearson correlation coefficient and root-mean-square errors (RMSE) between the actual data y_i and the predicted value \hat{y}_i .

4.4. ARIMA, LSTM and Seq2Seq Models

This section describes other models we developed to benchmark Transformer-based model.

ARIMA A univariate ARIMA model is used as a baseline. It treats the time dependent ILI ratios as a univariate time series that follows a fixed dynamic. Each week’s ILI ratio is dependent on previous p weeks’ observations and previous q weeks’ estimation errors. We selected the order of ARIMA model using AIC and BIC to balance model complexity and generalization. We used ARIMA(3, 0, 3) and a constant trend to keep the model parsimonious. The model is formulated in the State Space modeling framework and trained with the first two-thirds of the dataset. The fitted parameters are then used on the full time series to filter hidden states and make four-step ahead predictions.

LSTM The LSTM model has a stack of two LSTM layers and a final dense layer to predict the multiple step ILI ratios directly. The LSTM layers encode sequential information from input through the recurrent network. The dense connected layer takes final output from the second LSTM layer and outputs a vector of size 4, which is equal to the number of steps ahead predictions. The two LSTM layers are 32 and 16 units respectively. A dropout rate 0.2 is applied to LSTM layers for regularization. Huber loss, Adam optimizer, and a learning rate of 0.02 are used for training.

Seq2Seq The tested Seq2Seq model has an encoder-decoder architecture, where the encoder is composed of a fully connected dense layer and a GRU layer to learn from the sequential input and to return a sequence of encoded outputs and a final hidden state. The decoder is of the same structure as input. The dense layer is of 16 units and the GRU layer is of 32 units. An attention mechanism is also adopted in this Seq2Seq model. Specifically, Bahdanau attention (Bahdanau et al., 2015) is applied on the sequence of encoder outputs at each decoding step to make next step prediction. Teacher forcing (Williams & Zipser, 1989) is utilized in decoder for faster convergence and to address instability. During training, the true ILI rate is used at current

time step as input for the next time step, instead of using output computed from the decoder unit. A dropout rate 0.2 is applied in all recurrent layers. We used Huber loss, Adam optimizer, and a learning rate of 0.02 for training.

5. Experiment

5.1. One-step-ahead Forecasting Using ILI Data Alone

In our first experiment, we tested whether our Transformer-based model could predict the ILI ratio one-week ahead from 10 weeks of historical datapoints. For evaluation, the trained global model performs one-step ahead prediction for each state using the testing data set. Pearson correlation and root-mean-square error (RMSE) values were calculated for each state.

We compared the Transformer’s performance with ARIMA, LSTM, and Seq2Seq with attention models. Table 1 summarizes the correlation coefficients and RMSEs for each method, as well as relative performance gain with respect to ARIMA method. The comparison suggests that deep learning models overall outperform ARIMA for both correlation and RMSE. Within the three deep learning approaches, the correlation coefficients are very similar with the Transformer-based model being slightly higher than LSTM and Seq2Seq with attention models. In terms of RMSE, the Transformer model outperforms both LSTM and Seq2Seq with attention models, with relative RMSE decrease of 27 % and 8.4 %, respectively. This analysis suggests that attention mechanisms contribute to forecasting performance, as Seq2Seq with attention and Transformer models outperform the plain LSTM model. Additionally, the Transformer shows better forecasting performance compared to Seq2Seq with attention model, suggesting that Transformer’s self-attention mechanism can better capture complex dynamical patterns in the data compared to the linear attention mechanism used in Seq2Seq. Interestingly, it’s worth noting that Transformer exhibits the best metrics for US-level ILI forecasting (Pearson correlation = 0.984 and RMSE = 0.3318). Since a single model is trained using data from all the states, it suggests that the model indeed can generalize various state-level patterns for country-level prediction.

5.2. One-step-ahead Forecasting Using Feature Vectors

We next tested whether our Transformer-based model can learn from multiple features (i.e., multivariate time series data) for ILI forecasting. In the United States, the flu season usually starts in early October and peaks between January and February. We hypothesized that week number is an informative signal for the model. Therefore we introduced “week number” as a time-indexed feature to the model. Additionally, we included the first and second order differences

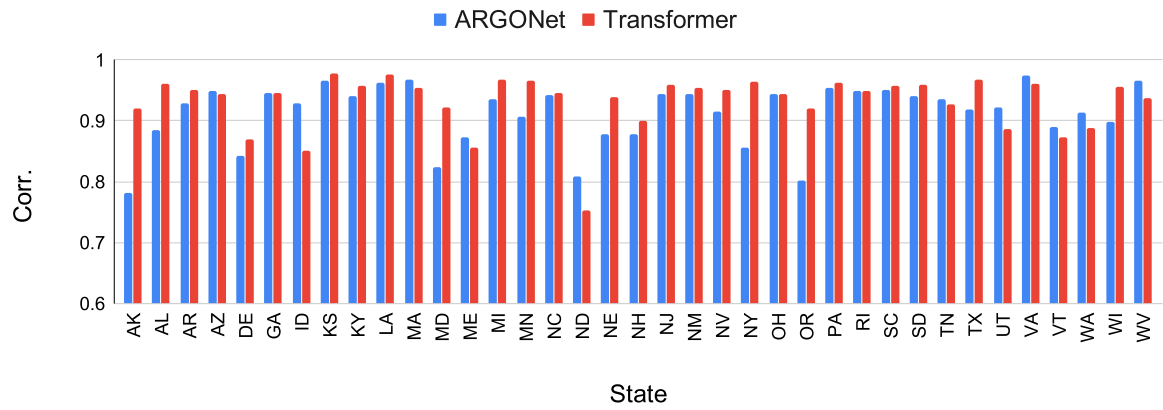


Figure 6. Pearson Correlation of ARGONet and transformer models.

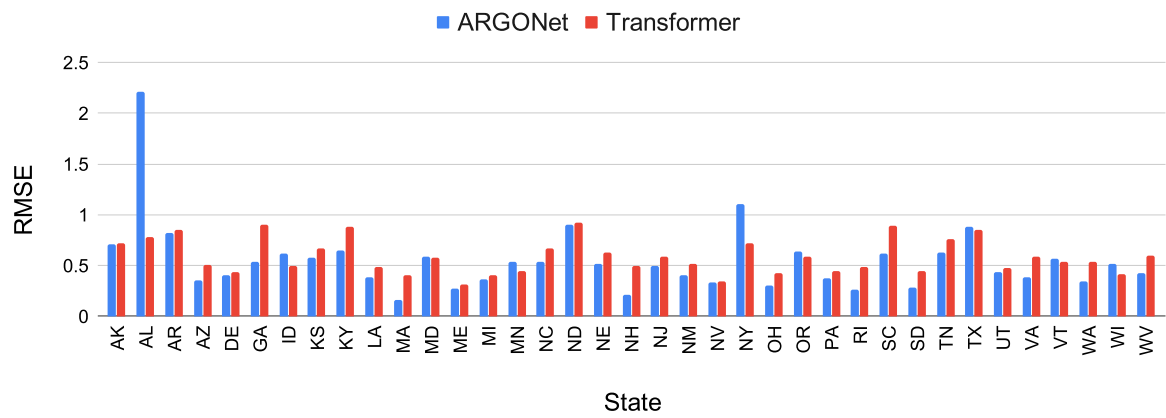


Figure 7. RMSE of ARGONet and transformer models.

Table 1. Summary of model performances with relative change with respect to baseline model.

Model	Pearson Correlation	RMSE
ARIMA	0.769 (+0 %)	1.020 (-0 %)
LSTM	0.924 (+19.9 %)	0.807 (-20.9 %)
Seq2Seq+attn	0.920 (+19.5 %)	0.642 (-37.1 %)
Transformer	0.928 (+20.7 %)	0.588 (-42.4 %)

of the time series as two explicit numerical features in the model.

Our results suggest that including these features improves model performance (mean Pearson correlation: 0.931, mean RMSE = 0.585). However, the improvement is not significant compared to the Transformer model using ILI data alone. This suggests that the additional features are likely to encode little new information to the model. That is, the introduced first and second order difference features are likely to be redundant if the Transformer-based model is able to rely on the self-attention mechanism to learn short and long-range dependencies from the ILI time series.

We compared our results with the ILI forecasting data by ARGONet (Lu et al., 2019), a state-of-the-art ILI forecasting model in the literature. Figure 6 and figure 7 show the correlation and RMSE values of ARGONet and our transformer results. Overall, the Transformer-based model performs equally with ARGONet, with the mean correlation slightly improved (ARGONet: 0.912, Transformer: 0.931), and mean RMSE value slightly degraded (ARGONet: 0.550, Transformer: 0.593).

5.3. Forecasting Using Time Delay Embedding

In this section, we tested whether the Transformer-based model can directly model phase space of a dynamical system. To that end, we constructed time delay embeddings (TDEs) from historical ILI data, since TDEs (with sufficient dimensionality) are topologically equivalent to the unknown phase space of dynamical systems. In other words, compared to ILI data, which are observed scalar variables, TDEs encode additional geometrical and topological information of the systems that governs the process of influenza infection and spreading. Therefore, using TDEs should provide richer information compared to scalar time series input.

To verify this hypothesis, we constructed time delay embeddings of dimension 2 to 32 from ILI data and applied transformer-based ILI forecasting using TDEs as features. Table 2 summarizes the forecasting metrics with different

TDE dimensions d . In all the experiments, we use $\tau = 1$ to construct TDEs. Varying TDE dimensionality does not significantly alter Pearson correlation coefficients. The RMSE value reached minimum with dimensionality of 8. This value is close to the optimal TDE dimensionality of 5 and 5-7 for forecasting chickenpox and measles (Sugihara & May, 1990).

Table 2. Performance of Time Delay Embeddings

Dimension	Pearson Correlation	RMSE
2	0.926	0.745
4	0.929	0.778
6	0.927	0.618
8	0.926	0.605
16	0.925	0.623
32	0.925	0.804

6. Conclusions

In this work, we presented a Transformer-based approach to forecasting time series data. Compared to other sequence-aligned deep learning methods, our approach leverages self-attention mechanisms to model sequence data, and therefore it can learn complex dependencies of various lengths from time series data.

Moreover, this Transformer-based approach is a generic framework for modeling various non-linear dynamical systems. As manifested in the ILI case, this approach can model observed time series data as well as phase space of state variables through time delay embeddings. It is also extensible and can be adapted to model both univariate and multivariate time series data with minimum modifications to model implementations.

Finally, although the current case study focuses on time series data, we hypothesize that our approach can be further extended to model spatio-temporal data indexed by both time and location coordinates. Self-attention mechanisms can be generalized to learn relations between two arbitrary points in spatio-temporal space. This is a direction we plan to explore in the future.

References

- Cdc fluview dashboard. <https://gis.cdc.gov/grasp/fluview/fluportaldashboard.html>.
- Araz, O. M., Bentley, D., and Muelleman, R. L. Using google flu trends data in forecasting influenza-like-illness related ed visits in omaha, nebraska. *The American Jour-*

-
- nal of Emergency Medicine*, 32(9):1016–1023, Sep 2014. ISSN 0735-6757.
- Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.
- Durbin, J. and Koopman, S. J. *Time Series Analysis by State Space Methods: Second Edition*. Oxford University Press, 2nd edition, 2012.
- Ginsberg, J., Mohebbi, M. H., Patel, R. S., Brammer, L., Smolinski, M. S., and Brilliant, L. Detecting influenza epidemics using search engine query data. *Nature*, 457(7232):1012–1014, 2009. ISSN 1476-4687.
- Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- Iuliano, A. D., Roguski, K. M., Chang, H. H., Muscatello, D. J., Palekar, R., Tempia, S., Cohen, C., Gran, J. M., Schanzer, D., Cowling, B. J., Wu, P., Kyncl, J., Ang, L. W., Park, M., Redlberger-Fritz, M., Yu, H., Espenhain, L., Krishnan, A., Emukule, G., van Asten, L., Pereira da Silva, S., Aungkulanon, S., Buchholz, U., Widdowson, M.-A., Bresee, J. S., and Network, G. S. I.-a. M. C. Estimates of global seasonal influenza-associated respiratory mortality: a modelling study. *Lancet (London, England)*, 391(10127):1285–1300, Mar 2018. ISSN 1474-547X.
- Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.
- Kondo, K., Ishikawa, A., and Kimura, M. Sequence to sequence with attention for influenza prevalence prediction using google trends. In *Proceedings of the 2019 3rd International Conference on Computational Biology and Bioinformatics, ICCBB '19*, pp. 1–7, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450376815.
- Lazer, D., Kennedy, R., King, G., and Vespignani, A. The parable of google flu: Traps in big data analysis. *Science*, 343(6176):1203–1205, 2014. ISSN 0036-8075.
- Liu, L., Han, M., Zhou, Y., and Wang, Y. Lstm recurrent neural networks for influenza trends prediction. In Zhang, F., Cai, Z., Skums, P., and Zhang, S. (eds.), *Bioinformatics Research and Applications*, pp. 259–264, Cham, 2018. Springer International Publishing. ISBN 978-3-319-94968-0.
- Lu, F. S., Hattab, M. W., Clemente, C. L., Biggerstaff, M., and Santillana, M. Improved state-level influenza nowcasting in the united states leveraging internet-based data and network approaches. *Nature Communications*, 10(1): 147, 2019. ISSN 2041-1723.
- McIver, D. J. and Brownstein, J. S. Wikipedia usage estimates prevalence of influenza-like illness in the united states in near real-time. *PLOS Computational Biology*, 10(4):1–8, 04 2014.
- Olson, D. R., Konty, K. J., Paladini, M., Viboud, C., and Simonsen, L. Reassessing google flu trends data for detection of seasonal and pandemic influenza: a comparative epidemiological study at three geographic scales. *PLoS computational biology*, 9(10):e1003256–e1003256, 2013. ISSN 1553-7358.
- Paul, M. J., Dredze, M., and Broniatowski, D. Twitter improves influenza forecasting. *PLoS currents*, 6: ecurrents.outbreaks.90b9ed0f59bae4ccaa683a39865d9117, Oct 2014. ISSN 2157-3999.
- Putri, W. C., Muscatello, D. J., Stockwell, M. S., and Newall, A. T. Economic burden of seasonal influenza in the united states. *Vaccine*, 36(27):3960 – 3966, 2018.
- Santillana, M., Zhang, D. W., Althouse, B. M., and Ayers, J. W. What can digital disease detection learn from (an external revision to) google flu trends? *American Journal of Preventive Medicine*, 47(3):341–347, Sep 2014. ISSN 0749-3797.
- Sugihara, G. and May, R. M. Nonlinear forecasting as a way of distinguishing chaos from measurement error in time series. *Nature*, 344(6268):734–741, 1990. ISSN 1476-4687.
- Takens, F. Detecting strange attractors in turbulence. In Rand, D. and Young, L.-S. (eds.), *Dynamical Systems and Turbulence, Warwick 1980*, pp. 366–381, Berlin, Heidelberg, 1981. Springer Berlin Heidelberg.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 5998–6008. Curran Associates, Inc., 2017.
- Venna, S. R., Tavanaei, A., Gottumukkala, R. N., Raghavan, V. V., Maida, A. S., and Nichols, S. A novel data-driven model for real-time influenza forecasting. *IEEE Access*, 7:7691–7701, 2019. ISSN 2169-3536.
- Williams, R. J. and Zipser, D. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

Yang, S., Santillana, M., and Kou, S. C. Accurate estimation of influenza epidemics using google search data via argo. *Proceedings of the National Academy of Sciences*, 112(47):14473–14478, 2015. ISSN 0027-8424.

Zhu, X., Fu, B., Yang, Y., Ma, Y., Hao, J., Chen, S., Liu, S., Li, T., Liu, S., Guo, W., and Liao, Z. Attention-based recurrent neural network for influenza epidemic prediction. *BMC Bioinformatics*, 20(18):575, 2019. ISSN 1471-2105.

深度 Transformer 模型在时间序列预测中的应用：流感盛行病例研究

Neo Wu, Bradley Green, Xue Ben, Shawn O'Banion

摘要

本文提出了一种新的时间序列预测方法。在这篇文章中，我们提出了一种全新的时间序列预测技术。众所周知，时间序列数据在许多科学和工程领域中都有广泛的应用。解读这些数据，并对其进行有效预测，一直是机器学习领域中的一个核心问题。我们的研究团队开发了一种新颖的方法，该方法采用了基于 Transformer 的机器学习模型，目的就是提高我们对时间序列数据的预测精度。这个方法利用自注意力机制，从时间序列数据中学习并理解其内在的复杂模式和动态变化。值得一提的是，这种预测框架具有很高的通用性。不论是对单变量还是多变量的时间序列数据，或者是时间序列嵌入，都可以应用这个框架进行预测。为了验证这个新方法的有效性，我们进行了一项案例研究，选择了流感样似症（ILI）的预测作为案例。研究结果表明，我们的方法所产生的预测结果与当前最先进的预测技术相比，具有极高的竞争力。

引言

季节性流感的全球流行给我们的健康和经济带来了沉重的压力。据统计，全球每年因流感致死的人数惊人地高达 291,000 至 646,000 人。在美国，疾病控制与预防中心（CDC）会利用他们的监测网络，每周发布一次有关流感症状的报告。虽然这种方式在追踪疾病流行的规模上发挥了重要作用，但由于数据收集和整理的时间，这些报告通常至少会有一周的延迟。因此，预测流感症状的活动就显得尤为重要，它能够帮助我们实时监测疾病的发展，以及为公共卫生机构分配资源、制定应对潜在大流行的计划和准备提供重要参考。

为了预测这些流感症状的时间序列数据，科学家们已经开发了各式各样的方法，从机械模型到统计和机器学习的方法，应有尽有。其中，机械模型主要是基于对疾病传播动态的理解。例如，像 SIR 这样的隔室方法就是常用的模拟疾病传播动态的工具。

统计和机器学习的方法则是借助真实数据去学习和掌握趋势与模式。其中，有一些广为流传的方法，例如自回归（AR）、自回归滑动平均（ARMA）以及自回归积分滑动平均（ARIMA）。除此之外，一些基于卷积和循环神经网络的深度学习方法也被开发出来，用于模拟流感症状数据。这些序列对齐的模型自然是处理时间序列数据的首选。然

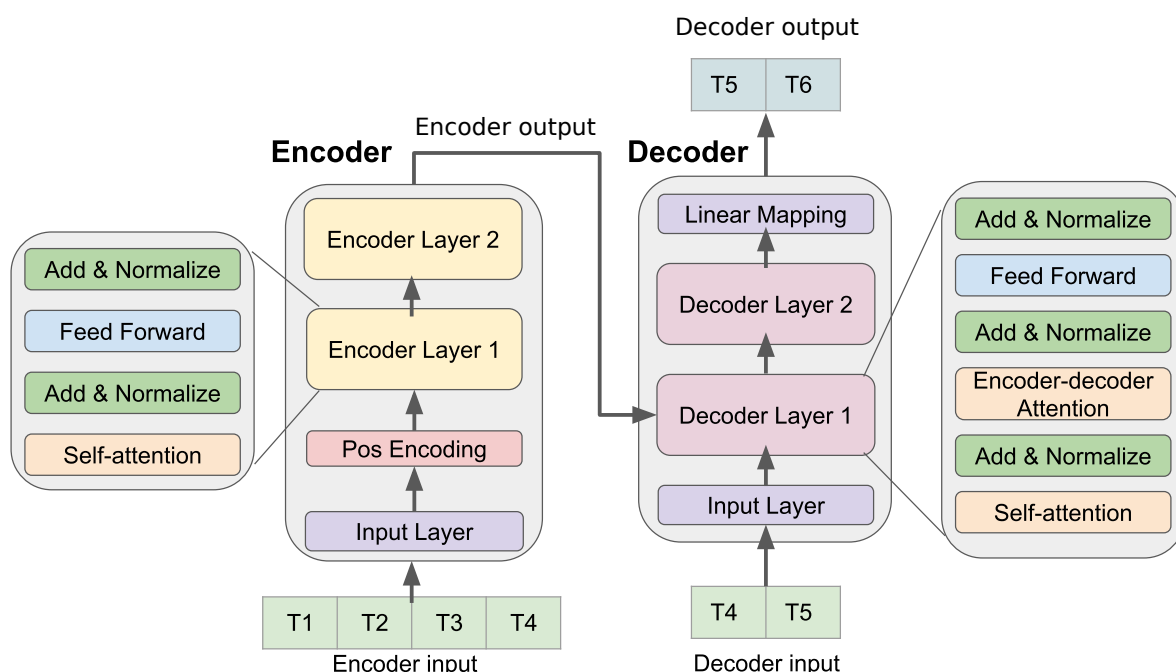


图 外 0-1 基于 Transformer 的预测模型架构

而，循环神经网络中存在梯度消失和梯度爆炸的问题，而卷积滤波器也有其局限性，这些都使得这些方法在处理序列数据中的长期和复杂关系时有所不足。

在我们的这项研究中，我们基于 Transformer 架构开发了一种新颖的时间序列预测方法。与序列对齐模型不同，Transformer 并不是按照顺序处理数据，而是一次性处理整个数据序列，并利用自我注意力机制去学习序列中的依赖关系。因此，基于 Transformer 的模型具有一定的潜力，能够处理那些对于序列模型具有挑战性的复杂时间序列数据。我们以流感症状预测为案例，展示了基于 Transformer 的模型在时间序列预测任务上的应用，结果表明，这种模型优于许多现有的预测技术。

相关工作

一些研究使用互联网数据，如 Google Trends、Twitter 和 Wikipedia 来预测流感样似症比例。Google 流感趋势（GFT）采用线性模型，利用预定义术语的 Google 搜索量来估计当前的流感样似症比例（“即时预测”）。尽管最初被认为是巨大的成功，但 GFT 在随后的几年中高估了峰值流感样似症数量。

其他研究将 GFT 与新的建模技术和额外信号结合起来。Lazer 等人提出了一种基于自回归（AR）的方法来扩展 GFT。Santillana 等人通过开发一种自动选择查询和更新流感样似症预测模型的模型，改进了 GFT。Araz 等人使用 GFT 数据和额外信号构建线性回归模型。Yang 等人开发了一种基于自回归的模型，利用 Google 搜索数据（“ARGO”）来估计流感流行病。ARGO 优于先前基于 Google 搜索的模型。最近，基于 ARGO 开发了一种新的集成模型（“ARGONet”）。这种新方法利用空间信息改进了模型，并实现了流感样似症预测的最新成果。

深度学习技术也被用于流感样似症预测。Liu 等人使用 Google Trends、气候、空气污染和病毒学监测数据训练了基于 LSTM 的模型来预测流感流行程度。Venna 等人开发了一个基于 LSTM 的多阶段模型，结合气候和时空调整因素进行流感预测。注意力机制也被应用于流感样似症预测。Zhu 等人开发了多通道 LSTM 神经网络，可以从不同类型的输入中进行学习。他们的模型使用注意力层将模型输出与输入序列关联起来，进一步提高了预测的准确性。Kondo 等人将序列到序列（“Seq2Seq”）模型与类似的注意力机制相结合，用于预测流感流行程度，并表明他们的方法优于 ARIMA 和基于 LSTM 的模型。

背景

流感和流感样似症（ILI）

流感是一种由流感病毒感染引起的常见传染病。每年流感影响多达 3500 万人，给健康和经济带来巨大负担。在美国，疾病控制与预防中心（CDC）在一个广阔的地理区域内协调一个报告网络。网络中的参与健康提供者报告显示流感样似症（ILI）症状的患者统计数据。ILI 症状通常定义为“发烧和咳嗽和/或喉咙痛”。ILI 比例被计算为患有 ILI 症状的患者数量与当周总门诊人数的比值。CDC 发布了美国和除佛罗里达州外的所有个别州的 ILI 比例。此外，州级的 ILI 比例还经过了对州人口的归一化处理。

状态空间模型

状态空间模型（State Space Models, SSM）广泛应用于动态系统。动态系统的演变受到不可观测的状态变量的控制。系统展现出由状态变量决定的可观测变量。SSM 已应用于生物学和金融等复杂系统的研究中。

状态空间模型同时建模状态变量和可观测变量。例如，广义线性状态空间模型可以用以下方程表示：

$$x_t = Z_t \alpha_t + \epsilon_t \quad \text{式（外 0-1）}$$

$$\alpha_{t+1} = T_t \alpha_t + R_t \eta_t, t = 1, \dots, n, \quad \text{式（外 0-2）}$$

其中， x_t 和 α_t 分别是时间索引的观测向量和状态向量。方程 外 0-1 称为观测方程，它类似于回归方程。它建模了可观测变量 x_t 与潜在状态变量 α_t 之间的关系。方程 外 0-2 是状态方程，具有自回归性质。它控制了状态变量随时间的演变。 ϵ_t 和 η_t 是创新项，通常被建模为高斯过程。在这一节中，我们简要提及了一些在 ILI 预测中常用的 SSM 模型。

隔室模型 隔室模型是 SSM 的一种特定形式，广泛应用于研究传染病。在隔室模型中，将人群分为不同的组（“隔室”）。每个组通过一个时间相关的状态变量来建模。一个著名的隔室模型是“易感-感染-康复”（Suscepted-Infected-Recovered, SIR）模型，该模型通过

以下常微分方程来描述系统的演变：

$$\begin{aligned}\frac{dS}{dt} &= -\frac{\beta IS}{N} \\ \frac{dI}{dt} &= \frac{\beta IS}{N} - \gamma I \\ \frac{dR}{dt} &= \gamma I\end{aligned}$$

在这个模型中，ILI 时间序列是系统的可观测变量： $ILI(t) = I(t)/(I(t) + S(t) + R(t))$ 。尽管最初是为了建模传染病而开发的，隔室模型已被应用于生态学和经济学等其他学科。虽然隔室模型很有用，但它们需要对微分方程的参数具有先验知识，并且缺乏在新观测数据下更新参数的灵活性。

ARIMA Box-Jenkins ARIMA（自回归差分移动平均）是另一种常用的建模动态系统的方法。ARIMA 模型对观测变量 x_t 进行建模，假设 x_t 可以分解为趋势、季节性和随机的成分。Box 和 Jenkins 的想法是对时间序列 x_t 进行差分操作，以消除趋势和季节性。得到的序列被视为平稳时间序列数据，并使用其滞后时间序列值的组合（“AR”）和滞后预测误差的移动平均值（“MA”）进行建模。ARIMA 模型通常由一个元组 (p, d, q) 来指定，其中 p 和 q 定义了 AR 和 MA 的阶数， d 指定了差分操作的阶数。

ARIMA 可以用 SSM 形式来表示，常见的 SSM 技术如滤波和平滑也可以应用于 ARIMA。然而，ARIMA 是一种“黑盒”方法，模型完全依赖于观测数据，没有对潜在系统状态的分析。

时间延迟嵌入 对于标量时间序列数据 x_t ，其时间延迟嵌入（TDE）是通过将每个标量值 x_t 嵌入到 d 维的时间延迟空间中形成的：

$$TDE_{d,\tau}(x_t) = (x_t, x_{t-\tau}, \dots, x_{t-(d-1)\tau})$$

对于任何非线性动力系统而言，延迟嵌入定理（Takens 定理）表明存在某个 (d, τ) -时间延迟嵌入，可以通过观测变量的延迟坐标近似恢复出原始状态变量（即“相空间”）的演化。在流感样似症预测的情况下，Takens 定理表明通过对 ILI 比例（即“观测变量”）构造的 $TDE_{d,\tau}$ ，可以近似描述由生物学和物理机制控制的潜在动力系统。

Sugihara 和 May 在开创性工作中首次探索了 TDE 在时间序列预测中的应用。他们展示了 TDE 可以通过对系统动力学的定性评估，无需了解底层机制，实现短期预测。他们开发了基于 TDE 的模型来预测水痘和麻疹的发病率，并将其与基于 AR 的方法进行比较。他们的分析表明，对于水痘病例的预测，TDE 模型与 AR 模型表现相当，而对于麻疹病例的预测，TDE 模型优于 AR 模型。

在状态空间模型（SSM）框架中，时间延迟嵌入是一种强大的工具，可以通过学习底层动力系统的几何和拓扑信息，将状态变量和观测数据联系起来，而无需理解系统的

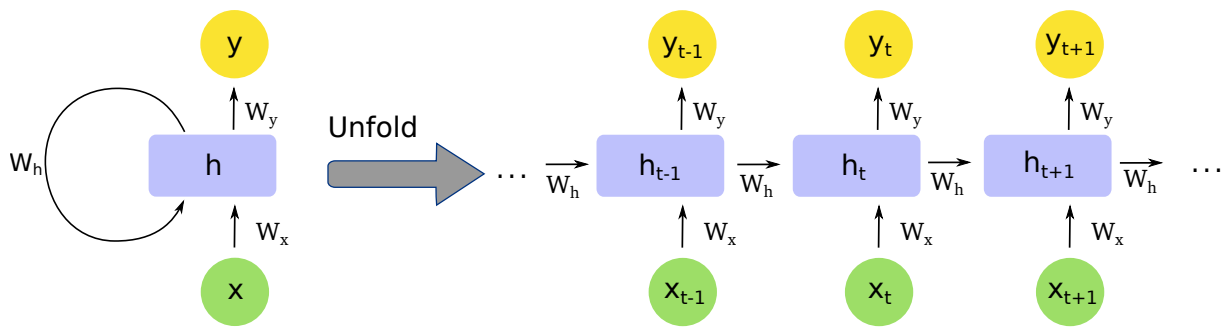


图 外 0-2 循环神经网络的折叠和展开表示

状态变量和相空间。尽管 TDE 具有令人惊叹的特性，但据我们所知，TDE 在机器学习模型中尚未得到广泛研究。

序列模型

许多现实世界的机器学习任务涉及不同类型的序列数据，包括自然语言文本、音频和视频、DNA 序列和时间序列数据。序列模型专门用于建模这种数据。在本节中，我们简要回顾几种常见的序列模型。

循环神经网络 与传统的前馈网络不同，循环神经网络（Recurrent Neural Networks, RNN）具有循环性质 - 它对每个输入 x_t 执行相同的函数，并且输出 y_t 依赖于输入 x_t 和前一个状态 h_{t-1} 。

如图 外 0-2 所示，简单的循环神经网络（Simple RNN）可以表示为：

$$h_t = \sigma(W_x x_t + W_h h_{t-1} + b_h)$$

$$y_t = \sigma(W_y h_t + b_y)$$

其中， x_t 是输入向量， h_t 是隐藏状态向量， y_t 是输出向量。 W 和 b 是学习参数， σ 是激活函数。

长短期记忆网络（LSTM） 虽然循环神经网络具有处理序列数据的内部记忆，但在处理长序列时会出现梯度消失和爆炸的问题。为了解决这个限制，专门开发了长短期记忆（Long Short-Term Memory, LSTM）网络。LSTM 使用三个门，包括输入门、遗忘门和输出门，来调节细胞之间的信息流动，防止梯度消失和爆炸。

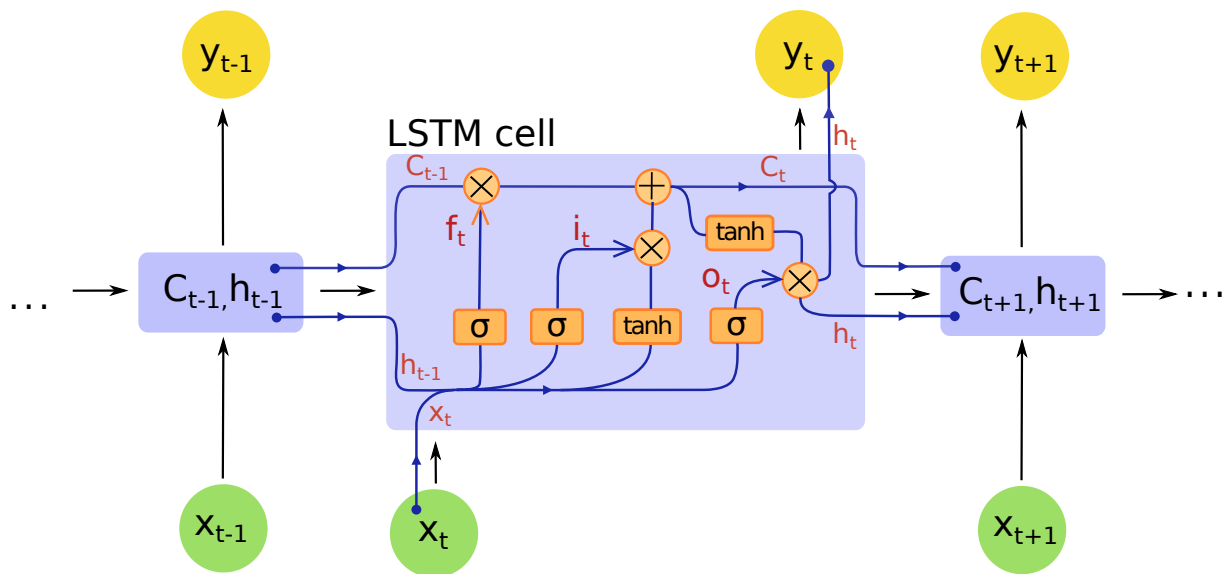


图 外 0-3 长短期记忆网络和 LSTM 单元

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$$y_t = \sigma(W_y[h_{t-1}, x_t] + b_y)$$

$$h_t = y_t * \tanh(C_t)$$

序列到序列模型 (Seq2Seq) 序列到序列 (Sequence-to-Sequence, Seq2Seq) 架构是为了处理输入和输出都是序列的机器学习任务而开发的。Seq2Seq 模型由三个组件组成，包括编码器 (Encoder)、中间向量 (Intermediate Vector) 和解码器 (Decoder)。编码器是一堆 LSTM 或其他循环单元。每个单元接收输入序列中的一个元素。编码器的最终隐藏状态被称为编码器向量或上下文向量，它编码了输入数据的所有信息。解码器也由一堆循环单元组成，并以编码器向量作为其第一个隐藏状态。每个循环单元计算自己的隐藏状态并产生一个输出元素。图 外 0-4 展示了 Seq2Seq 架构。

Seq2Seq 在语言翻译任务中得到了广泛应用。然而，它在处理长句子时性能下降，因为它无法充分将长序列编码到中间向量中（即使使用 LSTM 单元）。因此，在编码器向量中往往会丢失长期依赖关系。

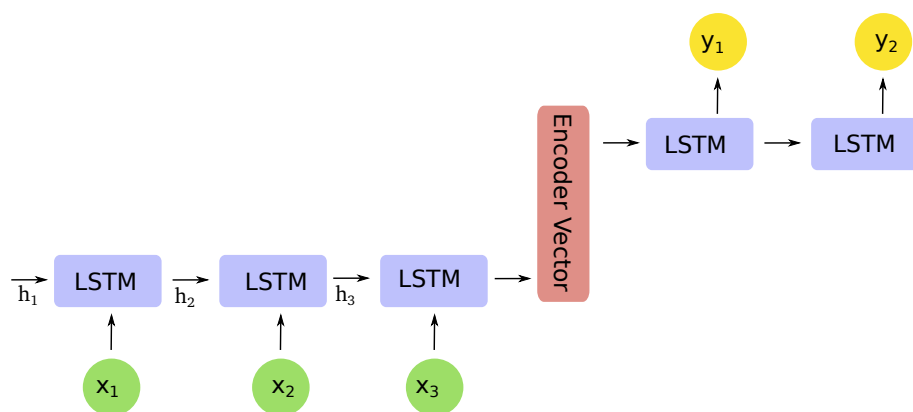


图 外 0-4 序列到序列模型架构

模型

问题描述

我们将 ILI 预测问题建模为一个监督机器学习任务。给定包含 N 个周数据点的时间序列 $x_{t-N+1}, \dots, x_{t-1}, x_t$, 对于 M 步预测, 监督机器学习模型的输入 X 为 $x_{t-N+1}, \dots, x_{t-M}$, 输出 Y 为 $x_{t-M+1}, x_{t-M+2}, \dots, x_t$ 。每个数据点 x_t 可以是一个标量或包含多个特征的向量。

数据

我们利用了来自 CDC 的 2010 年至 2018 年的国家和州级历史 ILI 数据。

为了生成带标签的数据集, 我们采用了固定长度的滑动时间窗口方法 (图 外 0-5), 构建用于模型训练和评估的 X, Y 对。在应用滑动窗口之前, 我们对所有数据进行最大最小值缩放, 使用训练数据集的最大和最小值。然后在缩放后的训练集上运行滑动窗口, 获得具有特征和标签的训练样本, 其中特征是前 N 个观测值, 标签是后 M 个观测值。测试样本也以相同的方式构建, 用于模型评估。训练和测试集的划分比例为 2:1。来自不同州的训练数据被连接起来形成全局模型的训练集。

Transformer 模型

模型架构

我们基于原始的 Transformer 架构构建了基于 Transformer 的 ILI 预测模型, 包括编码器和解码器层。

编码器 编码器由输入层、位置编码层和四个相同的编码器层组成。输入层通过一个全连接网络将输入的时间序列数据映射为一个维度为 d_{model} 的向量。这一步对于模型使用多头注意力机制至关重要。使用正弦和余弦函数的位置编码对时间序列数据中的顺序信息进行编码, 通过将输入向量与位置编码向量进行逐元素相加来实现。得到的向量被馈

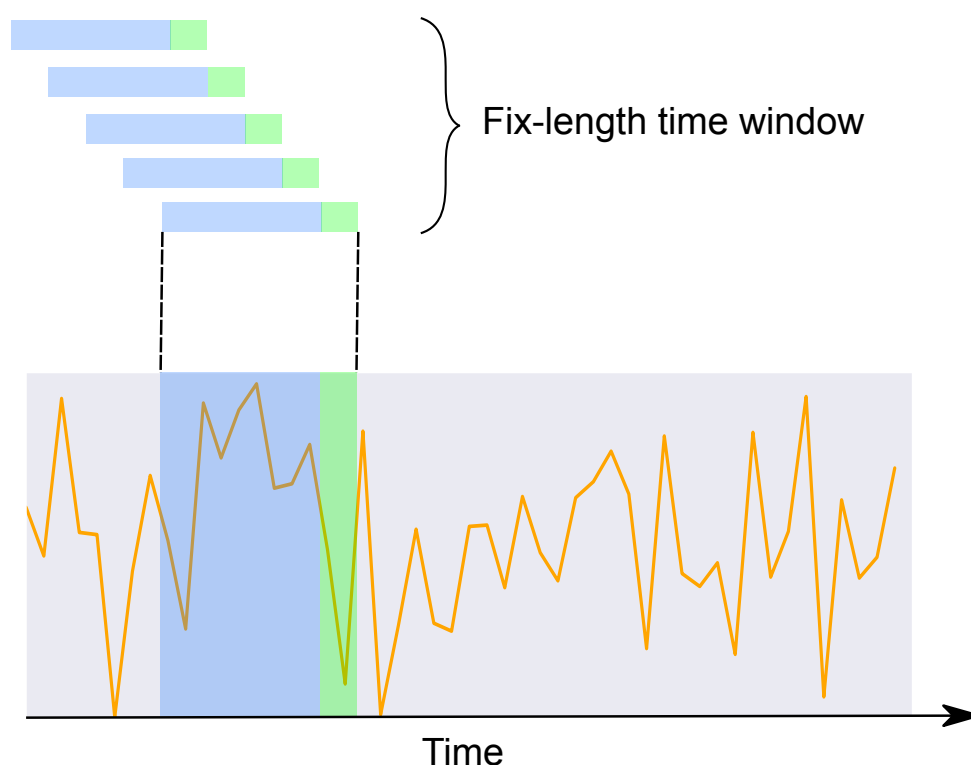


图 外 0-5 使用滑动时间窗口构建训练样本和测试样本

送到四个编码器层。每个编码器层包括两个子层：自注意力子层和全连接前馈子层。每个子层之后都有一个归一化层。编码器产生一个 d_{model} 维的向量，用于馈送给解码器。

解码器 我们采用了与原始 Transformer 架构相似的解码器设计。解码器由输入层、四个相同的解码器层和一个输出层组成。解码器的输入从编码器输入的最后一个数据点开始。输入层将解码器输入映射为一个维度为 d_{model} 的向量。除了每个编码器层中的两个子层外，解码器还插入第三个子层，对编码器输出应用自注意力机制。最后，有一个输出层将最后一个解码器层的输出映射到目标时间序列。在解码器中，我们采用前瞻遮挡和解码器输入与目标输出之间的一位偏移，以确保对时间序列数据点的预测仅依赖于先前的数据点。

训练

训练数据和批处理 在典型的训练设置中，我们训练模型从过去 10 个周的数据预测未来 4 周的 ILI 比例。也就是说，给定编码器输入 $(x_1, x_2, \dots, x_{10})$ 和解码器输入 (x_{10}, \dots, x_{13}) ，解码器的目标是输出 (x_{11}, \dots, x_{14}) 。我们使用前瞻遮挡来确保模型只会在目标数据之前的数据点上应用注意力。也就是说，在预测目标 (x_{11}, x_{12}) 时，遮挡确保注意力权重仅作用在 (x_{10}, x_{11}) 上，以防止解码器从解码器输入中泄露关于 x_{12} 和 x_{13} 的信息。训练时使用大小为 64 的小批量数据。

优化器 我们使用 Adam 优化器，其中 $\beta_1 = 0.9$ ， $\beta_2 = 0.98$ ， $\epsilon = 10^{-9}$ 。我们使用自定义的学习率和以下调度表进行训练：

$$lrate = d_{\text{model}}^{0.5} * \min(\text{step_num}^{0.5}, \\ \text{step_num} * \text{warmup_steps}^{-1.5})$$

其中 $\text{warmup_steps} = 5000$ 。

正则化 我们对编码器和解码器中的三种子层（自注意力子层、前馈子层和归一化子层）都应用了 dropout 技术。每个子层的 dropout 率为 0.2。

评估

在评估中，我们使用固定长度的滑动窗口构建了带有标签的测试数据。经过训练的 Transformer 模型进行一步预测。我们计算实际数据 y_i 和预测值 \hat{y}_i 之间的皮尔逊相关系数和均方根误差（RMSE）进行评估。

ARIMA、LSTM 和 Seq2Seq 模型

本节介绍了我们开发的其他模型，用于对比 Transformer-based 模型的性能。

ARIMA 我们使用 ARIMA 模型作为基准模型。ARIMA 模型将时变的 ILI 比例视为一个遵循固定动态的单变量时间序列。每周的 ILI 比例取决于前 p 周的观测值和前 q 周的估计误差。我们使用赤池信息准则（AIC）和贝叶斯信息准则（BIC）来选择 ARIMA 模型的阶数，以在模型复杂性和泛化性之间取得平衡。我们选择了 ARIMA(3, 0, 3) 的阶数，并添加了一个常数趋势，以保持模型的简洁性。该模型基于状态空间建模框架进行训练，使用数据集的前三分之二进行训练。然后，使用拟合的参数对完整的时间序列进行滤波，进行四步的预测。

LSTM LSTM 模型由两个 LSTM 层的堆叠和一个最后的全连接层组成，直接预测多步的 ILI 比例。LSTM 层通过循环网络从输入中编码顺序信息。全连接层接收来自第二个 LSTM 层的最终输出，并输出一个大小为 4 的向量，与预测的步数相对应。两个 LSTM 层分别具有 32 个和 16 个单元。为了正则化，我们在 LSTM 层中应用了 0.2 的丢弃率。训练时使用 Huber 损失、Adam 优化器和学习率为 0.02。

Seq2Seq Seq2Seq 模型采用了一种与 Transformer 模型类似的编码器-解码器架构，不过它并没有使用自注意力机制。它的编码器和解码器都由两层 LSTM 构成。编码器的作用是将输入的序列转化为一个上下文向量，然后解码器利用这个上下文向量以及前一个

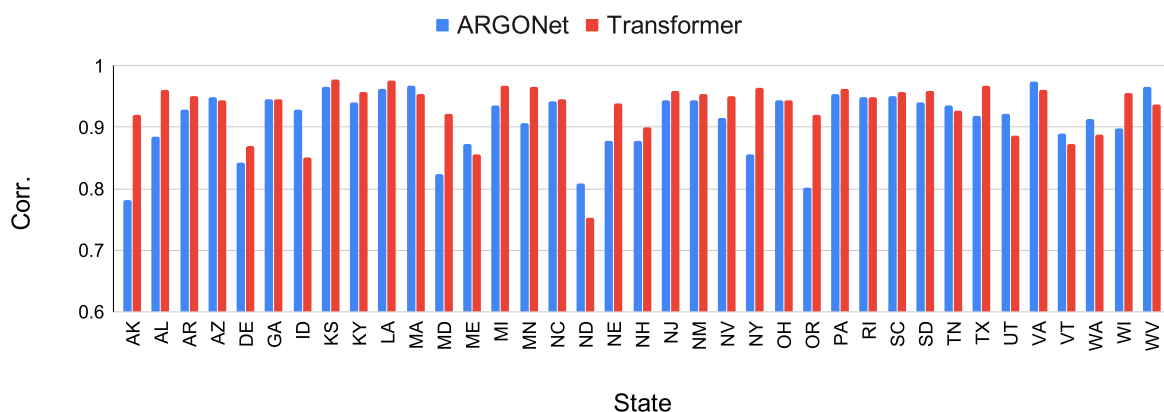


图 外 0-6 ARGONet 和 Transformer 模型的皮尔逊相关系数

输出逐步生成目标序列。在训练这个 Seq2Seq 模型的过程中，我们采用了 Huber 损失、Adam 优化器和 0.02 的学习率。

Seq2Seq 模型的主要用途是作为基准模型，用来与基于 Transformer 的模型进行对比和性能评估。

Seq2Seq 我们测试的 Seq2Seq 模型也采用了编码器-解码器架构，编码器部分由一个全连接的稠密层和一个 GRU（门控循环单元）层组成，其主要任务是从输入的序列中提取和学习特征，最终返回一组编码的输出和最后一个隐藏状态。解码器的结构与编码器部分基本相同。在这个模型中，稠密层包含 16 个单元，而 GRU 层则包含 32 个单元。值得一提的是，这个 Seq2Seq 模型还巧妙地融入了注意力机制。具体来说，解码器在每一步的解码过程，都会利用 Bahdanau 注意力机制处理编码器的输出序列，以便更好地进行下一步的预测。解码器还使用了一种名为“Teacher forcing”的技术，这有助于模型更快地收敛，同时解决了不稳定性问题。这种技术的基本思路是，在训练过程中，我们会将当前时间步的真实 ILI 比例作为下一个时间步的输入，而不是使用解码器单元计算出的输出。此外，为了防止过拟合，所有的循环层中都采用了 0.2 的丢弃率。在训练这个模型的过程中，我们使用了 Huber 损失函数、Adam 优化器和 0.02 的学习率。

实验

仅使用 ILI 数据进行一步预测

在我们的第一个实验中，我们测试了我们基于 Transformer 的模型是否能够从历史数据中预测一周后的 ILI 比例。为了评估模型，训练好的全局模型使用测试数据集进行一步预测。计算了每个州的皮尔逊相关系数和均方根误差（RMSE）。

我们将 Transformer 模型的性能与 ARIMA、LSTM 和带有注意力机制的 Seq2Seq 模型进行了比较。表 外 0-1 总结了每种方法的相关系数和 RMSE，以及相对于 ARIMA 方法的性能提升。比较表明，深度学习模型在相关系数和 RMSE 方面整体优于 ARIMA。

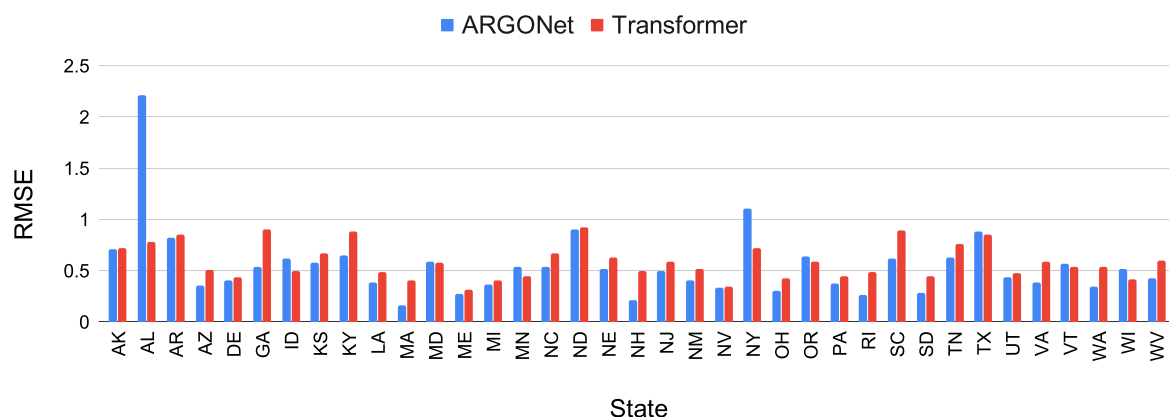


图 外 0-7 ARGONet 和 Transformer 模型的均方根误差 (RMSE)

在这三种深度学习方法中，相关系数非常接近，Transformer 模型略高于 LSTM 和带有注意力机制的 Seq2Seq 模型。在 RMSE 方面，Transformer 模型的性能优于 LSTM 和带有注意力机制的 Seq2Seq 模型，相对的 RMSE 减小分别为 27% 和 8.4%。这个分析表明，注意力机制对预测性能起到了贡献，因为带有注意力机制的 Seq2Seq 和 Transformer 模型优于纯 LSTM 模型。此外，与带有注意力机制的 Seq2Seq 模型相比，Transformer 模型显示出更好的预测性能，这表明 Transformer 的自注意机制能够更好地捕捉数据中的复杂动态模式。值得注意的是，Transformer 在美国的 ILI 预测中显示出最佳的指标（皮尔逊相关系数 = 0.984，RMSE = 0.3318）。由于单个模型使用来自所有州的数据进行训练，这表明该模型确实能够推广各个州的模式，用于国家级的预测。

表 外 0-1 模型性能总结及相对于基准模型的变化

模型	皮尔逊相关系数	RMSE
ARIMA	0.769 (+0 %)	1.020 (-0 %)
LSTM	0.924 (+19.9 %)	0.807 (-20.9 %)
Seq2Seq+ 注意力机制	0.920 (+19.5 %)	0.642 (-37.1 %)
Transformer	0.928 (+20.7 %)	0.588 (-42.4 %)

使用特征向量的逐步预测

接下来，我们测试了基于 Transformer 的模型是否能够从多个特征（即多变量时间序列数据）中学习进行 ILI 预测。在美国，流感季节通常从 10 月初开始，并在 1 月和 2 月之间达到高峰。我们假设周数是模型的一个信息量丰富的信号。因此，我们将“周数”作为一个与时间相关的特征引入模型中。此外，我们还在模型中包括时间序列的一阶和二阶差分作为两个显式的数值特征。

我们的结果表明，包含这些特征可以提高模型的性能（平均皮尔逊相关系数：0.931，平均 RMSE=0.585）。然而，与仅使用 ILI 数据的 Transformer 模型相比，改进并不显著。这表明这些额外的特征可能对模型提供了较少的新信息。也就是说，如果基于 Transformer 的模型能够依赖自注意力机制从 ILI 时间序列中学习短期和长期依赖关系，那么引入的一阶和二阶差分特征很可能是多余的。

我们将结果与 ARGONet 的 ILI 预测数据进行了比较，ARGONet 是文献中的一种先进的 ILI 预测模型。图 外 0-6 和图 外 0-7 显示了 ARGONet 和我们的 Transformer 模型的相关系数和 RMSE 值。总体而言，基于 Transformer 的模型与 ARGONet 的性能相当，平均相关系数略有改善（ARGONet: 0.912，Transformer: 0.931），平均 RMSE 值略有下降（ARGONet: 0.550，Transformer: 0.593）。

采用时间延迟嵌入法进行预测分析

在这一部分，我们针对基于 Transformer 的模型能否直接对动力系统的相空间进行建模进行了一番实验探究。我们采用了历史的 ILI 数据，以此构建出时间延迟嵌入 (TDEs)。值得注意的是，当 TDEs 的维度足够多时，它在拓扑结构上就能等同于那些未知的动力系统相空间。

换句话说，相对于我们观测到的标量变量 ILI 数据，TDEs 所编码的内容包含了更丰富的几何和拓扑信息，这些信息是控制流感感染和传播过程的系统必需的。因此，如果我们采用 TDEs，就能获取到比标量时间序列输入更丰富的信息。

为了检验这个理论，我们从 ILI 数据中构建了从 2 到 32 维的时间延迟嵌入，并以此作为特征，进行基于 Transformer 的 ILI 预测。在表 外 0-2 中，我们列出了在不同 TDE 维度 d 下的预测指标结果。在所有的实验中，我们均采用了 $\tau = 1$ 来构建 TDEs。

通过观察我们发现，改变 TDE 的维度并不会对皮尔逊相关系数产生显著影响。然而，在维度增加的过程中，RMSE 值先是降低，后又趋于稳定，其中在维度为 8 时达到了最小值。这个结果与预测水痘和麻疹所使用的最优 TDE 维度相近，那时的最优维度分别为 5 和 5-7。

表 外 0-2 时间延迟嵌入的性能

维度	皮尔逊相关系数	RMSE
2	0.926	0.745
4	0.929	0.778
6	0.927	0.618
8	0.926	0.605
16	0.925	0.623
32	0.925	0.804

结论

在这项研究里，我们提出了一种全新的，基于 **Transformer** 的时间序列预测技术。相比其他依赖于序列对齐的深度学习方法，我们的方法可以更有效地利用自注意力机制来捕捉和模拟序列数据的特性，从而能在时间序列数据中学习并理解各种长度和复杂度的依赖关系。

值得一提的是，我们所采用的基于 **Transformer** 的方法构成了一个通用框架，能够处理和建模各种非线性动力系统。这一点在 **ILI** 案例中得到了清晰的展示：这种方法能够通过时间延迟嵌入来建模观察到的时间序列数据和状态变量的相空间。此外，这种方法还具备较强的可扩展性，无论是对单变量还是多变量的时间序列数据建模，都只需对模型实现进行微小的修改即可。

最后，我们想要强调的是，尽管目前的案例研究主要集中在时间序列数据上，我们坚信我们的方法有着更广泛的应用潜力，可以进一步扩展到建模以时间和位置坐标为索引的时空数据。这是因为自注意力机制有着极强的泛化能力，可以用来学习时空空间中任意两点之间的关系。这也将是我们未来研究的重要方向。

北 京 邮 电 大 学

本科毕业设计（论文）开题报告

学院	计算机学院（国家示范性软件学院）	专业	智能科学与技术	班级	2019211315
学生姓名	张梓靖	学号	2019211379	班内序号	27
指导教师姓名	王纯	所在单位	计算机学院（国家示范性软件学院）	职称	高级工程师
设计(论文)题目	（中文）一种基于工作量的 Serverless 计算自动伸缩算法的设计与实现				
	（英文）Design and Implementation of Workload-based Auto-scaling Algorithm for Serverless Computing				

一、 选题背景及意义

Serverless 计算是近年来比较流行的一种云计算模型，它提供了一种无服务器的计算方式，可以大大降低企业运维成本，提高系统的弹性和可用性。但是，随着业务的发展，Serverless 服务的负载情况也会发生变化，有时会出现突发流量的情况，导致系统资源不足，影响服务的正常运行。为了解决这一问题，我们计划开发一种基于工作量的 Serverless 计算自动伸缩算法，使用前沿机器学习方法如 LSTNet、TPA-LSTM 等，根据 Serverless 服务的历史负载情况，进行时间序列预测，从而实现更加精准、节约、高效的自动伸缩。

Serverless 是云计算的一种设计思想，它的特点在于，不需要用户持续维护服务器、操作系统以及代码运行所需的基本环境，而是将这些前置需求部署到云端。这样，用户可以更专注于开发和部署应用，而不必担心底层基础架构的管理和维护。

Serverless 计算的主要特点包括：

- 按需执行：代码会在请求到来时被触发执行，而不是持续运行，从而有效降低计算成本。
- 无服务器：用户不需要维护服务器，也不需要关心服务器的配置和管理。
- 可扩展性：在需要时自动增加或减少计算资源，以应对流量高峰期或突发事件。
- 简单开发：在 Serverless 计算中，用户只需要关注应用的业务逻辑，而无需

关心底层架构。这样可以大大简化开发过程，并且能够更快地完成应用的部署和扩展。

二、 研究的基本内容

虽然自动伸缩技术在 Serverless 计算中发挥了重要作用，但它目前也存在一些问题。

一方面是自动伸缩算法的精确度。目前自动伸缩算法大多是基于某些预定义的性能指标或阈值来决定扩展或缩减计算资源，但这些指标和阈值并不能完全反映应用的实际需求。因此，一些自动伸缩算法可能会导致过度扩展或过度缩减计算资源，从而影响应用的性能和可用性。

另一方面，自动伸缩技术的可靠性也是一个问题。目前自动伸缩技术的可靠性并不完美。折衷不完美是两方面的，一种是伸缩过程的不完美，比如可能会导致系统故障或自动伸缩失败，从而影响应用的性能和可用性。另一种是伸缩策略的不完美，例如依赖于历史周期的伸缩，能否良好适应突发情况，是需要考虑的。

本项目主要针对前者，即算法的精确度尝试进行改进。此外，目前基于机器学习或传统学习、预测方法的自动伸缩算法往往能将周期性把握得很好，但对于长期趋势的把握可能不够。实际上随着业务的发展，有可能在存在日、周、月度周期的同时，还存在整体的业务上升趋势，从而急切需要一个更有适应性的模型。

三、 研究方法及措施

我们试图探索一种方法，能够不但捕捉到容器伸缩的时间周期，也能捕捉到伸缩随着业务发展的长期趋势。

在更广泛的意义上，这一项目也是在推动机器学习技术在自动伸缩领域的应用，为相关行业提供更多的选择和便利。同时，通过对 Serverless 计算负载情况的深入研究和分析，我们还可以为相关行业提供更为专业的建议和解决方案。

同时，本项目还会研究如何推进具体技术的落地。因为目前机器学习算法在落地过程中，往往发现有推理速度慢、资源消耗高、运行条件苛刻的情况。所以我们会研究如何通

过一些先进的技术，例如深度学习编译，模型优化等技术，来推进算法的落地。

对于一个发展期的业务，不但存在短期的周期性（例如日、周访问量会有明显的周期），往往还存在长期的上升趋势（例如这个月 DAU 等指标，比上个月要高出一定数值，或是一定比例），所以长期上有可能表现出线性或者指数型增长。

在这种情况下，由于时间序列数据具有周期性和长期上升趋势，因此可以使用深度学习模型或者结合深度学习模型和线性模型的方法来进行预测。例如 LSTNet/TPA-LSTM/TCN 模型。

对于一个稳定期的业务，可能只存在短期的周期性，而长期，例如月、年的尺度上，由于主要是存量市场，用户增长和流失的速率基本平衡，运营也相对稳定，导致流量变化不显著。由于时间序列数据只具有短期的周期性，长期尺度上流量变化不显著，因此可以使用基于时间特征的线性回归模型或者 ARIMA 模型来进行预测。

四、 研究工作的步骤与进度

2023.1.1 ~ 2023.2.10 完成领域内容调研，论文对应部分撰写。

2023.2.28~2023.4.15 完成相关研究，设计程序。

2023.4.16~2023.4.30 进行设计评估和比较分析。

2023.5.1~2023.5.15 论文整体撰写。

五、 主要参考文献

[1] Kubernetes Authoritative guide version 4, author: Zheng Gong, Zhihui Wu, Xiulong Cui, Jianyong Yan.

[2] Docker technology introduction, author: Baohua Yang.

[3] Kubernetes docs: <https://kubernetes.io/docs/home/>

[4] OpenFaaS docs: <https://docs.openfaas.com/>

[5] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti and et al., “Cloud programming simplified: A berkeley view on serverless computing,” arXiv preprint arXiv:1902.03383, 2019.

[6] Laszlo Toka, Gergely Dobreff, Balazs Fodor and Balazs Sonkoly, “Adaptive AI-based auto-scaling for Kubernetes,” in 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet

Computing (CCGRID). IEEE, 2020, pp. 559-608.

[7] BINGO Hong, 时间序列预测方法总结: <https://zhuanlan.zhihu.com/p/67832773>

指导教师签字



日期

2023 年 1 月 1 日

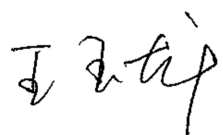
注：可根据开题报告的长度加页。

北 京 邮 电 大 学

2023 届本科毕业设计（论文）中期进展情况检查表

学院	计算机学院（国家示范性软件学院）		专业	智能科学与技术	
学生姓名	张梓靖	学号	2019211379	班级	2019211315
指导教师姓名	王纯	所在单位	计 算 机 学 院 （国家示范性软件学院）	职称	高级工程师
设计(论文)题目	（中文）一种基于工作量的 Serverless 计算自动伸缩算法的设计与实现				
	（英文）Design and Implementation of Workload-based Auto-scaling Algorithm for Serverless Computing				


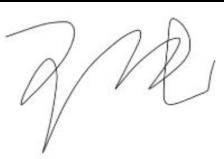
目前 已 完 成 任 务	<p>主要内容：</p> <p>简要而言：</p> <ol style="list-style-type: none">1. 目前已经实现了使用 SA、AR、ARIMA 和 SARIMA 方法进行预测。2. 已经能够将预测算法对接到 OpenFaaS 等环境中使用。3. 可以继续尝试基于 CNN 的机器学习方法。 <p>下面是详细内容。</p> <p>首先，我们来看已经实现的预测算法，其中包括了四种不同的方法：SA、AR、ARIMA 和 SARIMA。SA 就是简单平均，我们用它作为一种 Baseline，从而确保我们的预测算法没有纰漏，因为一般情况下效果应该优于简单平均。AR (Autoregressive) 方法则是一种基于时间序列的预测算法，通过分析历史数据来预测未来的趋势。ARIMA (Autoregressive Integrated Moving Average) 和 SARIMA (Seasonal ARIMA) 方法是基于 AR 方法的扩展，增加了对数据的差分和季节性特征的处理，以更准确地预测未来的趋势。这些算法在各种领域都得到了广泛应用，例如经济预测、天气预报、股票预测等。也同样适用于我们的 OpenFaaS 等环境。</p> <p>其次，我们已经能够将这些预测算法对接到 OpenFaaS 等环境中使用。OpenFaaS 是一种轻量级、可扩展的 Serverless 平台，可以让开发者更轻松地创建和部署函数。</p> <p>在前面的进展报告中，我们提到了已经能够将预测算法对接到 OpenFaaS 等环境中使用。这种部署方式是将预测算法封装成函数，然后通过 API 调用来使用它们。尽管这种方式具有高度的灵活性和可扩展性，但是它与具体的 FaaS 平台耦合在一起，限制了部署的灵活性和可移植性。为了解决这个问题，我们将预测算法封装为单独的服务，与具体的 FaaS 平台解耦。这种服务化的部署方式可以让预测算法在不同的平台上使用，并且具有更好的可移植性和可扩展性。</p> <p>具体来说，我们可以使用 Docker 等容器技术将预测算法封装为服务，然后通过 Kubernetes 等容器编排工具来部署和管理这些服务。这种部署方式可以让我们更好地控制计算资源，同时也具有更好的可伸缩性和容错性。此外，我们还可以使用 Istio 等服务网格技术来管理不同服务之间的通信和流量控制，从而更好地保证服务的可用性和稳定性。</p> <p>为了进一步提高部署的灵活性和可扩展性，我们还可以通过 FaaS 框架提供的外部接口控制伸缩。比如说，OpenFaaS 提供了 Function Label，我们可以通过打标签的方式控制 Function 的基础配置，例如最小副本数。这样来根据实际负载情况自动调整服务的实例数量。这种方式可以让我们更好地响应负载变化，同时也可以降低成本和资源浪费。</p>
	是否符合任务书要求进度 是

尚 需 完 成 的 任 务	<p>我们可以继续尝试基于 RNN 的机器学习方法。这类算法广泛应用于图像处理、自然语言处理等领域。与传统的时间序列预测算法相比，RNN 具有更强的自适应性和非线性特征提取能力。</p> <p>我们还需要对代码进行重构，以便更好地部署。</p>		
	<p>是否可以按期完成设计（论文） 是 <input checked="" type="checkbox"/> 否 <input type="checkbox"/></p>		
存 在 问 题 和 解 决 办 法	存 在 问 题	<ol style="list-style-type: none"> 1. 没有公开的长期（大于一年）精确（粒度小于一分钟）的 Serverless 实际运行情况数据集 2. 也没有条件在生产环境接触到这样的数据集。 3. 还应该对比更多的算法，选择其中最为适合的。 	
	拟 采 取 的 办 法	<ol style="list-style-type: none"> 1. 使用同类数据（进行模拟 2. 降低预测的精度，更关注全局的趋势性，而短期的快速扩容交给简单阈值方法。 	
指 导 教 师 签 字		日 期	2023 年 4 月 14 日
检 查 小 组 评 分 及 意 见	<p>评分： 23（总分：25）</p> <p>通过</p> <div style="text-align: center;">  </div> <p>组长签字： 2023 年 4 月 14 日</p>		



注：可根据长度加页。



北 京 邮 电 大 学



教师指导本科毕业设计（论文）记录表

学院	计算机学院（国家示范性软件学院）	专业	智能科学与技术	班级	2019211315
学生姓名	张梓靖	学号	2019211379	班内序号	27
指导教师姓名	王纯	职称	高级工程师		
<p>第 1—2 周记录：</p> <p>给予学生与毕业设计相关的论文，撰写任务书，给学生转达毕设相关信息与时间安排要求。</p>					
指导教师签字		日期	2022 年 11 月 30 日		
<p>第 3—4 周记录：</p> <p>指导学生调研论文，并在算法设计上给予建议，对整个算法提出需求和目标，指导学生编写开题报告。</p>					
指导教师签字		日期	2022 年 12 月 6 日		

注：每 2 周指导内容记录在一个表格中，双面打印。

学院	计算机学院（国家示范性软件学院）	专业	智能科学与技术	班级	2019211315
学生姓名	张梓靖	学号	2019211379	班内序号	27
指导教师姓名	王纯	职称	高级工程师		
<p>第5—6周记录：指导学生对算法整体结构和思想写出计划书，并对计划书中的疑惑给出一定的建议，修改计划书中不合理的地方。</p>					
指导教师签字		日期	2023 年 3 月 14 日		
<p>第7-8周记录：</p> <p>指导学生根据计划书中的方案编写代码，解答学生提出的疑惑，对设计方面做出反馈和修改意见。</p>					
指导教师签字		日期	2023 年 3 月 28 日		

学院	计算机学院（国家示范性软件学院）	专业	智能科学与技术	班级	2019211315
学生姓名	张梓靖	学号	2019211379	班内序号	27
指导教师姓名	王纯	职称	高级工程师		
第 9-10 周记录：					
指导学生根据计划书中的方案编写代码，解答学生提出的疑惑，对设计方面做出反馈和修改意见。					
指导教师签字		日期	2023 年 4 月 7 日		
第 3-4 周记录：					
根据算法的实际效果，给出相关合理建议，使算法能够收敛，并且得到较优的结果。					
指导教师签字		日期	2023 年 4 月 18 日		

学院	计算机学院（国家示范性软件学院）	专业	智能科学与技术	班级	2019211315
学生姓名	张梓靖	学号	2019211379	班内序号	27
指导教师姓名	王纯	职称	高级工程师		
<p>第 13—14 周记录：</p> <p>根据算法的实际效果，给出相关合理建议，使算法能够收敛，并且得到较优的结果。</p>					
指导教师签字		日期	2023 年 4 月 29 日		
<p>第 15-16 周记录：</p> <p>指导学生编写毕业论文，完成查重工作。对毕业论文的格式、内容提出要求。并对毕业论文提出修改意见。</p>					
指导教师签字		日期	2023 年 5 月 16 日		

第 17 周记录:

指导学生完成毕业论文，指导学生完成毕业论文的答辩，完成整个毕业设计工作。

指导教师签字



日期

2023 年 5 月 23 日